NO. 47 APRIL 1982

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £1.80

MICRO™

THE 6502/6809 JOURNAL

6809 Feature

BASIC Utilities for Apple and PET

Votrax Interface

Inexpensive Mass Storage Techniques

THE CHIEFTAIN™ 5¼-INCH WINCHESTER HARD DISK COMPUTER

SO ADVANCED IN SO MANY WAYS . . . AND SO COST-EFFECTIVE . . . IT OBSOLETES MOST OTHER SYSTEMS AVAILABLE TODAY AT ANY PRICE.

● HARD DISK SYSTEM CAPACITY

The Chieftain series includes 5¼- and 8-inch Winchesters that range from 4- to 60-megabyte capacity, and higher as technology advances. All hard disk Chieftains include 64-k memory with two serial ports and DOS69D disk operating system.

● LIGHTNING ACCESS TIME

Average access time for 5¼-inch Winchesters is 70-msec, comparable to far more costly hard disk systems. That means data transfer ***ten-times faster*** than floppy disk systems.

The Chieftain Computer Systems:

Here are the Chieftain 6809-based hard disk computers that are destined to change data processing . . .

☐ **CHIEFTAIN 95W4**
4-megabyte, 5¼-inch Winchester with a 360-k floppy disk drive (pictured).

☐ **CHIEFTAIN 95XW4**
4-megabyte, 5¼-inch Winchester with a 750-k octo-density floppy disk drive.

☐ **CHIEFTAIN 98W15**
15-megabyte, 5¼-inch Winchester with a 1-megabyte 8-inch floppy disk drive.

☐ **CHIEFTAIN 9W15T20**
15-megabyte, 5¼-inch Winchester with a 20-megabyte tape streamer.

● 2-MHZ OPERATION

All Chieftains operate at 2-MHz, regardless of disk storage type or operating system used. Compare this to other hard disk systems, no matter ***how*** much they cost!

● DMA DATA TRANSFER

DMA data transfer to-and-from tape and disk is provided for optimum speed. A special design technique eliminates the necessity of halting the processor to wait for data which normally transfers at a slower speed, determined by the rotational velocity of the disk.

● RUNS UNDER DOS OR OS-9

No matter which Chieftain you select . . . 5¼- or 8-inch floppy, or 5¼- or 8-inch Winchester with tape or floppy back-up . . . they ***all*** run under DOS or OS-9 with ***no need*** to modify hardware or software.

● UNBOUNDED FLEXIBILITY

You'll probably never use it, but any Chieftain hard disk system can drive up to 20 other Winchesters, and four tape drives, with a single DMA interface board!

● SMOKE SIGNAL'S HERITAGE OF EXCELLENCE

This new-generation computer is accompanied by the same ***Endurance-Certified*** quality Dealers and end-users all over the world have come to expect from Smoke Signal. And support, software selection and extremely competitive pricing are very much a part of that enviable reputation.

20-Megabyte Tape Streamer Back-Up Option

Available with all Chieftain hard disk configurations. This cartridge tape capability provides full 20-megabyte disk back-up in less than five minutes with just one command, or copy command for individual file transfers. Transfers data tape-to-disk or disk-to-tape. Floppy back-up is also available in a variety of configurations.

Write or call today for details (including the low prices) on the Chieftain Series. . . and on dealership opportunities

SMOKE SIGNAL BROADCASTING®

31336 VIA COLINAS
WESTLAKE VILLAGE, CA 91362
TEL (213) 889-9340

Name ______________________

Company ______________________

Address ______________________

City ____________ State ______ Zip ______

Telephone ( ) ______________________

NO. 47 APRIL 1982

MICRO™

THE 6502/6809 JOURNAL

DEPARTMENTS

I/O ENHANCEMENTS

HARDWARE

6809

BASIC AIDS


Boston
Minneapolis
Houston
San Francisco
apple fest
82

About the Cover

FLOWER	PERIOD	COLOR
Zinnia	Aug.-Sept.	Multi
Tulip	May-June	Various
Marigold	Jul.-Aug.	Gold
Iris	May-June	White-Purple
Crocus	April	Purple,yellow,white
Peony	June	Pink,white,fuscMie
Aster	Aug.-Sept.	Multi

This month's cover launches MICRO into spring with colorful tulips. A microcomputer could be used to plan garden planting. Given the characteristics of the plants — their growing seasons, stature, flower color, etc. — the program would assist in planting for best balance.

The 'spring' theme of the cover also relates directly to the editorial theme of the issue — the 6809. This is truly the spring of the 6809, as well!

Cover photo: Betsey Bolton
Lowell, Massachusetts

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

MICRO Editorial

Support the 6809!

Personal computers make the transfer of information easier by removing unnecessary barriers between minds. The result is more than just added convenience. According to information theory (and common sense), if it is easier for people not to have information than to have it, they won't have it. How many times have you known that you could obtain a piece of information if you were willing to make the trip to the library? Chances are you stayed home and remained ignorant. Personal computers offer us the possibility of lives that are "barrier-free" with respect to information.

"Barrier-free" is a term used to describe building designs that don't lock out the handicapped. Nearly everyone has been handicapped at one time or another by not having access to the right information at the right time. What may be called the "personal computing movement" generates such intense enthusiasm because we are dimly aware that making the flow of information barrier-free can offer human society opportunities for advancement greater than any known before in history.

Even so, the personal computing industry has been responsible for creating some new barriers as it removes the old ones. The familiar "Tower of Babel" analogy aptly describes the problem — those who can't use each new computer language far outnumber those who can. In the Biblical story, the Tower of Babel was a joint effort by all of humanity to build a structure that would attain the heavens. To prevent this, God inflicted "Babel" on his presumptuous children so that only small groups could solve problems in common with the aid of that powerful tool, language. As humanity still strives to create that great tower of common understanding, it is still language, our greatest resource, that is our most imposing barrier.

Those of us who are professionally involved in the growth of personal computers have a responsibility to make this technology as barrier-free as possible. Barriers that have become familiar — between the Apple world and the TRS-80 world, for example — are not in anyone's long-term best interest. Nor are they going to be meaningful much longer. Radio Shack has announced the Model 16, which will incorporate the 68000, the same chip rumored to be part of the next-generation Apple. But even when systems use the same CPU, it can be extremely difficult to alter code written for one configuration to run on another unless sytem transportability has been a major design consideration from the start.

There is a microprocessor available now that can play a significant role in removing barriers between systems. The 6809 microprocessor, designed at Motorola, removes obstacles to transportability that the 6502, for all its virtues, created. Hardware considerations required a fixed page zero location in the 6502, making it very difficult to alter 6502 software written for a specific system to run on another 6502 configuration. The 6809's Direct Page Register, however, permits the software itself to establish page zero in the process of adapting to specific system configurations. The result: position-independent code.

One of MICRO's primary concerns is to promote the removal of barriers to software transportability. We are pleased, therefore, to feature the 6809 processor in this issue, which includes a discussion of the 6809 *vis-a-vis* the 6502 by Mssrs. Walker and Whiteside of Motorola.

I would like to conclude by taking this opportunity to introduce myself to the readers of MICRO. As Senior Editor, I hope to help MICRO become an even more effective information interchange between serious computerists. If you have any comments or ideas, write or call me at MICRO. Or reach me at 71535,231 on the CompuServe network.

Laurence Kepple

AIM User Device Arbiter

by Joel Swank

Expand the AIM's user input and output ports up to 83 devices each with the User Device Arbiter.

AIM User Device Arbiter requires:
AIM-65

One of AIM's strongest features is tbe user I/O port, system device "U". With this user book you can interface a wide variety of devices to the AIM and they will work with all AIM firmware. Unfortunately only one input and one output device can be available at a time. Since I use several devices on the user port, remembering the device driver addresses and manually changing the user vectors was inconvenient. To relieve this problem, I wrote the User Device Arbiter (UDA).

UDA separates the AIM user port into as many as 83 sub-devices. Each subdevice is represented by a one-character code. When I specify "U" in response to tbe IN= or OUT= prompt, the UDA receives control and displays tbe prompt DEVICE=. If I enter tbe one-character sub-deivce code, tbe open routine for tbat device is then executed. Any subsequent calls to the user port are sent to the device driver through the secondary user vector in the UDA.

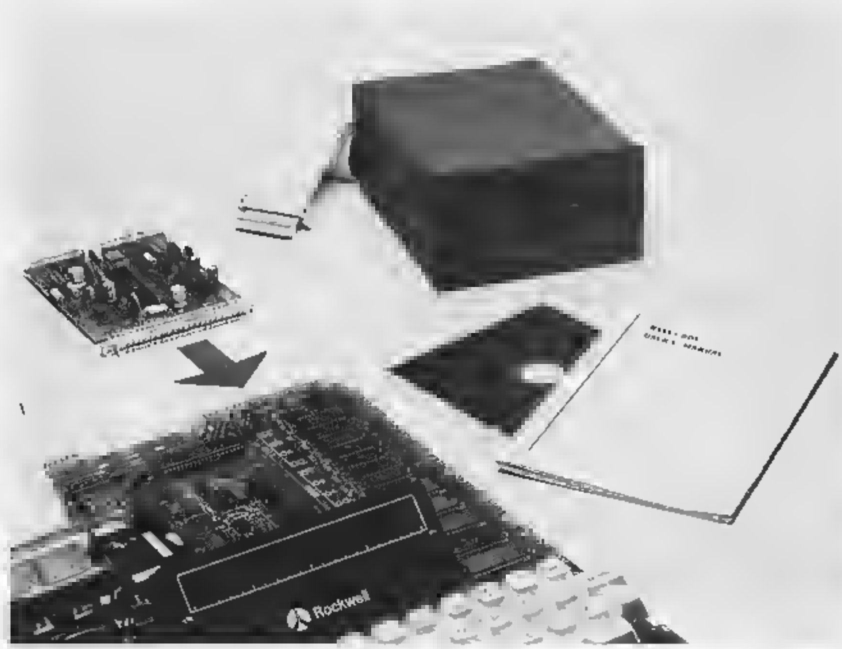
UDA is a simple, table-driven routine. There are two logically identical routines, one for input and one for output. The Arbiter routines are only executed when they are entered with the carry flag clear (open call). The response to the DEVICE= prompt is used as a search argument for the device table, which is a list of device codes and device driver routine addresses. The driver routines are the same routines whose addresses would normally be stored in the user vectors. The tables must be terminated with a zero. If a device code is not found in the table, the error message UNKNOWN DEVICE is displayed and the DEVICE= prompt re-issued. When the device code is

```
                  ;    UDA : THE AIM USER DEVICE ARBITER
                  ;
                  ;     FUNCTION:
                  ;         TO SELECT AMONG MULTIPLE DEVICES FOP
                  ;          I/O VIA THE AIM USER PORT.

                  ; AIM USEP VIA ADDRESSES

                  UDRB   =$A000
                  UDDRB  =$A002
                  UPCR   =$A00C
                  UIFR   =$A00D
                  UIER   =$A00E

                  ;   AIM SUBROUTINES

                  TTYTST =$E842          ;TEST FOR TTY MODE
                  BLANK  =$E83E          ;SEND SRACE TO D/R
                  OUTPUT =$E97A          ;ACCUM TO D/P
                  EQUAL  =$E7D8          ;'=' TO THE D/P
                  REDOUT =$E973          ;READ KBD WITH ECHO
                  CRLOW  =$EA13          ;CR LF TO D/P
                  COMIN  =$E1A1          ;AIM RE-ENTRY

                  UIN    =$108           ;AIM USER I/O VECTORS

                  ;   OUTSIDE ADDRESSES

                  VICIN  =$962F          ;VIC-20 I/O
                  VICOT  =$864B
                  DISKIN =$93C3          ;DISK I/O DRIVERS
                  DISKOT =$937A
                  BUFFIN =$8B96          ;BUFFER MANAGER I/O DRIVERS
                  BUFFOT =$8841

                       *=$8000

                  ;   ROUTINE TO INITIALIZE THE USER I/O VECTORS

8000  A0 03       INITAL LDY #3
8002  89 E8 80    INILUP LDA VECS,Y      ;INIT USER I/O VECTORS
8005  99 08 01           STA UIN,Y
8008  88                 DEY
8009  10 F7              BPL INILUP
800B  4C A1 E1           JMP COMIN
800E              -------------------
800E              ARBITER ROUTINES

800E              ;   ENTRY FOR USER INPUT

800E  B0 22       USERI  BCS JMPIN       ;ALREADY OPEN

8010  20 AF 80    GETI   JSR GETDEV      ;INPUT DESIRED DEVICE
8013  A0 00              LDY #0          ;PREPARE TO SEARCH TABLE
8015  BE D1 B0    UDILUP LDX DTABI,Y     ;END OF TABLE?
8018  F0 18              BEQ NODEVI      ;YES, ERROR
801A  D9 D1 80           CMP DTABI,Y     ;MATCH?
801D  F0 05              BEQ MOVADI      ;YES, DISPATCH IT
801F  C8                 INY
8020  C8                 INY             ;NO, BUMP TO NEXT
8021  C8                 INY
8022  D0 F1              BNE UDILUP      ;TRY AGAIN

8024  C8          MOVADI INY             ;FOUND - BUMP TO ADDRESS
8025  B9 D1 B0           LDA DTABI,Y     ;MOVE ADDRESS TO VECTOR
8028  8D 12 81           STA IVEC
802B  B9 D2 B0           LDA DTABI+1,Y
802E  8D 13 81           STA IVEC+1
8031  18                 CLC

8032  6C 12 B1    JMRIN  JMP (IVEC)      ;EXECUTE DEVICE DRIVER
```

(Continued)

DOS FOR AIM-65®: $499.00 COMPLETE!

INSTANT EASY ACCESS TO YOUR AIM-65 PROGRAMS!
MANIPULATE OBJECT FILES AT THE TOUCH OF A KEY.
NEW BYTE-DOS DISC OPERATING SYSTEM.
153K BYTES ON-LINE. INEXPENSIVELY!



BYTE-DOS COMMANDS

- **FORMAT** - Prepares New Disc
- **BAD SECTOR SCAN** - QC's Disc
- **DUMP** - RAM File to Disc
- **LOAD** - Disc File to Ram
- **LIST** - Prints Disc File Names
- **ERASE** - Eliminate a File
- **UNERASE** - File Recovery
- **RENAME** - Rename a File
- **COPY** - Copy 1 File
- **TRANSFER** - Copy Complete Disc
- **KOMPRESS** - Optimizes Disc Space

AND MORE!

IT'S EASY AND FUN TO USE!

BYTE MICROSYSTEMS
CORPORATION
1477 ELKA AVENUE, SAN JOSE, CA 95129 408/446/0559

- *BYTE-DOS OPERATES WITH THE AIM-65 MONITOR, BASIC, ASSEMBLER, EDITOR, PL/65 AND FORTH. ACCESS IS THROUGH PROGRAM I/O AND F1-F3 KEYS. BYTE-DOS SUPPORTS ONE OR TWO 5¼" OR 8" FLOPPYS!*
- *BYTE-DOS $499 SYSTEM INCLUDES DISC CONTROLLER CARD, TEAC FD-50A DISC DRIVE (153K FORMATTED BYTES), BYTE-DOS SOFTWARE SYSTEM, CABLES, CONNECTORS, MANUALS. COMPLETE! JUST PLUG IT TOGETHER AND USE!*

SEND ME THE FULL STORY!

Name ______________________
Company ______________________
Address ______________________
City __________ State ______ Zip ________
Telephone ______________ ☐ home ☐ work

☐ I own an AIM-65.
☐ I'm thinking of buying an AIM-65.
☐ Send full BYTE-DOS Data.

found in the table, the succeeding two bytes are moved to the secondary user vectors. Subsequent calls to the device "U" vectors with carry flag set (I/O call) are directed through these secondary vectors.

My device tables contain three input devices and four output devices. Included in the assembly is the driver for my BASE 2 printer. The drivers for my disk, buffer manager, and VIC-20 parallel link, are located elsewhere. Devices can easily be added by inserting their device codes and driver routine addresses in the tables. To avoid selecting the wrong device, have each open routine display a message that identifies which device was selected.

Included at the beginning of UDA is a routine that initializes the user I/O vectors with the addresses of the arbiter routines. Execute this routine only once after UDA is loaded. UDA has no effect on AIM's restriction of having only one input and one output device open at a time.

The author may be contacted at 25730 Beach Dr., Rockaway, OR 97136.

```
8035  20 C3 80  NODEVI JSR DIVERR       ;ERROR MSG
8038  4C 10 80         JMP GETI         ;RETRY


803B                   ;   ENTRY FOR USER OUTPUT

803B  B0 22     USERO  BCS JMPOT        ;ALREADY OPEN



803D  20 AF 80  GETO   JSR GETDEV       ;INPUT DESIRED DEVICE
8040  A0 00            LDY #0           ;PREPARE TO SEARCH TABLE
8042  BE DB 80  UDOLUP LDX DTABO,Y      ;END OF TABLE?
8045  F0 IB            BEQ NODEVO       ;YES, ERROR
8047  D9 DB 80         CMP DTABO,Y      ;MATCH?
804A  F0 05            BEQ MOVADO       ;YES, DISPATCH IT
804C  C8               INY
804D  C8               INY              ;NO, BUMP TO NEXT
804E  C8               INY
804F  D0 F1            BNE UDOLUP       ;TRY AGAIN

805I  C8        MOVADO INY              ;FOUND - BUMP TO ADDRESS
8052  B9 DB 80         LDA DTABO,Y      ;MOVE ADDRESS TO VECTOR
8055  8D 14 81         STA OVEC
8058  B9 DC 80         LDA DTABO+I,Y
805B  8D 15 81         STA OVEC+1
805E  18               CLC

805F  6C I4 8I  JMPOT  JMP (OVEC)       ;EXECUTE DEVICE DRIVER

8062  20 C3 80  NODEVO JSR DIVERR       ;ERROR MSG
8065  4C 3D 80         JMP GETO         ;RETRY

8068                   ;BASE 2 INTERFACE FOR THE AIM 65
8068                   ;USES THE B PORT OF THE USER 6522 VIA


8068                   ;    ENTRY TO USE THE BASE 2 THROUGH THE
8068                   ;      AIM USER OUTPUT VECTOR.

8068  B0 33     BASEOT BCS UBAS          ;BRANCH ON OUTPUT CALL
806A  A0 0F            LDY #BASMSG-LITS;DISPLAY 'BASE-2'
806C  20 C5 80         JSR PMSG


806F                   ;     SUBROUTINE TO INITIALIZE THE VIA
806F  A9 FF     BASINT LDA #$FF         ;ALL BITS OUTPUT
8071  8D 02 A0         STA UDDRB
8074  AD 0C A0         LDA UPCR
8077  29 0F            AND #$0F
8079  09 A0            ORA #$A0         ;SET AUTO PULSE MODE
807B  8D 0C A0         STA UPCR
807E  A9 00            LDA #0           ;SEND A NULL TO GET THINGS
                                         STARTED
8080  8D 00 A0         STA UDRB
8083  A2 00            LDX #0
8085  BD 03 81  BASLUP LDA INITS,X      ;LOOP TO SEND
8088  F0 06            BEQ CKTERM       ;PARMS TO BASE2
808A  20 A2 80         JSR BASOUT
808D  E8               INX
808E  D0 F5            BNE BASLUP
8090  20 42 E8  CKTERM JSR TTYTST       ;TTY MODE?
8093  F0 04            BEQ DIS          ;YES
8095  A9 37            LDA #55          ;NO, ENABLE AUTO LF
8097  D0 09            BNE BASOUT
8099  A9 38     DIS    LDA #56          ;DISABLE AUTO LF
809B  D0 05            BNE BASOUT

809D                   ;  CHARACTER FROM USER OUTPUT COMES HERE

809D  68        UBAS   PLA
809E  C9 FF            CMP #$FF         ;IGNORE AIM NULL CODES
80A0  F0 0C            BEQ BRET

80A2                   ;     SUBROUTINE TO SEND I CHARACTEP TO THE BASE2

80A2  48        BASOUT PHA
80A3  AD 0D A0  BOTLUP LDA UIFR         ;GET VIA STATUS
80A6  29 I0            AND #$I0         ;IS PRINTER READY?
80A8  F0 F9            BEQ BOTLUP       ;NO, WAIT
80AA  68               PLA
80AB  8D 00 A0         STA UDRB         ;YES, SEND CHARACTER
80AE  60        BRET   RTS
80AF                   ;   GETDEV ; INPUT DEVICE CODE FROM CONSOLE

80AF  20 I3 EA  GETDEV JSR CRLOW        ;NEW LINE
80B2  A0 08            LDY #DEVMSG-LITS;PROMPT 'DEVICE='
80B4  20 C5 80         JSR PMSG
```

```
80B7  20 D8 E7          JSR EQUAL
80BA  20 73 E9          JSR REDOUT      ;GET REPLY
80BD  48                PHA



80BE  20 3E E8          JSR BLANK       ;SEND SPACE
80C1  68                PLA
80C2  60                RTS

80C3              ;   DIVERR : DISPLAY ERROR MESSAGE

80C3  A0 00       DIVERR LDY #ERRMSG-LITS

80C5              ;   PMSG : MESSAGE WRITER

80C5  B9 EC 80    PMSG   LDA LITS,Y     ;GET A CHAR
80C8  F0 06              BEQ PDUN       ;QUIT ON NULL
80CA  20 7A E9           JSR OUTPUT     ;SEND IT
80CD  C8                 INY
80CE  D0 F5              BNE PMSG
80D0  60          PDUN   RTS

80D1              ;   DATA TABLES

80D1              ;   TABLE OF INPUT DEVICES

80D1              DTABI  =*
80D1  44                 .BYT 'D'
80D2  C3 93              .WOR DISKIN
80D4  56                 .BYT 'V'
80D5  2F 86              .WOR VICIN
80D7  42                 .BYT 'B'
80D8  96 88              .WOR BUFFIN
80DA  00                 .BYT 0

80DB              ;   TABLE OF OUTPUT DEVICES

80DB              DTABO  =*
80DB  44                 .BYT 'D'
80DC  7A 93              .WOR DISKOT
80DE  56                 .BYT 'V'
80DF  48 86              .WOR VICOT
80E1  42                 .BYT 'B'
80E2  41 88              .WOR BUFFOT
80E4  50                 .BYT 'P'
80E5  68 80              .WOR BASEOT
80E7  00                 .BYT 0

80E8              ;   USER I/O VECTOR INITS

80E8  0E 80       VECS   .WOR USERI
80EA  3B 80              .WOR USERO

80EC              ;   MESSAGE TABLE

80EC              LITS   =*
80EC  55 4E       ERRMSG .BYT 'UNKNOWN '
80F4  44 45       DEVMSG .BYT 'DEVICE',0
80F6  00
80FB  42 41       BASMSG .BYT 'BASE 2 ',0
80FD  00

80FE              ; TABLE OF INIT PARMS FOR BASE2
80FE              ; 96 CPL, 8 LPI, 88 LPP,
80FE              ; AUTO FF 4 LINES UP

8103  1B          INITS  .BYT 27,50,27,84,88,27,57,4
8104  32
8105  1B
8106  54
8107  58
8108  1B
8109  39
810A  04
810B  1B                 .BYT 27,58,27,98,18,27,0
810C  3A
810D  1B
810E  62
810F  12
8110  1B
8111  00

8112              ;   SECONDARY USER I/O VECTORS
8112              ;     MUST BE IN RAM

8112  00 00       IVEC   .WOR 0
8114  00 00       OVEC   .WOR 0
```

MICRO

OSI COMPATIBLE PRODUCTS

56K 2-MHz Ultra Low Power CMOS Static Memory Board MEM-56K $850

Partially Populated Boards (Specify address locations required) ... **MEM-48K $750**
MEM Board uses the new 2K-Byte Wide Static RAM chips which are 2716 EPROM compatible. Any 2K byte memory segment can be populated with RAM or EPROM (or left empty for use of Address Space by another board). Fully expandable to any memory size you will ever need. No special addressing requirements, just solder in extra sockets

	MEM-32K $550
	MEM-24K $450
	MEM-16K $350
	MEM- 8K $250
	MEM- 4K $200
Extra 2K RAM Memory Chip	$24
Optional Parallel Printer Port	-P $120
Optional Calendar/Clock Software available in EPROM)	-T $ 25
Both options (Disk software mods provided for use of 6522 VIA on printer).	-PT $125

EXAMPLE USES:

C4P & C8P: Expansion to **4K RAM** of Basic workspace.
Parallel Printer Port — Reserve Serial Port for MODEM
Calendar/Clock Displaying on unused portion of screen
Space for 5.75K of **Enhanced System Monitor EPROMS.**

All of this on 1 Board, using only one of your precious slots. Software for Enhanced System Monitor capabilities is continuously being developed and improved. As new EPROM Monitors are available, you may upgrade to them for any price differential plus a nominal $10 exchange fee. Another possibility is to fill any portion of the memory with Basic Programs in EPROM for **Power-on Instant Action.** This custom EPROM programming service is available at $25 per 2716 (Includes EPROM). Extra copies at $15 for each EPROM.

C4P-MF & C8P-DF:
Memory expansion to **48K.**
Add 6K **Memory** above BASIC for special software requirements.
Parallel Printer Interface and/or **Displaying Calendar/Clock.**
Add 1.75 K **Enhanced System Monitor ROM.**

C3: Up to **56K of Memory** Expansion — can be addressed for **Multiuser.** (Optionally, each user can have his own **Dedicated Printer Port**).

C1P,C4P & C8P FLOPPY DISC CONVERSIONS:

Memory/Floppy Board (Includes MI48P1 ROM)	**MEM F-16K $450**
C1P-600 Board Adapter & Cable	**A600/48 $ 50**
Additional Memory/Printer/Times	**(See MEM Board Prices)**
5¼" Drive/Case/Power Supply & Cable to MEMF Board	**FD5 $399**

IEE-488 INTERFACES AND SOFTWARE:

The General Purpose Instrumentation Bus (GPIB Controller interface is available for all OSI Computers. Machine code GPIB Drivers are linked to Basic to provide easy control of IEEE-488 instruments which is equal to the best of Hewlett-Packard Controllers and far superior to most others. Basic Commands for Serial Poll, Parallel Poll, IFC Clear, full Local/Remote Control, Respond to SRQ Interrupts, Send Trigger, do Formatted Input/Output, Direct Memory Input/Output and MORE. Interface includes IEEE-488 Ribbon Cable/Connector

GPIB Controller Interface for C2, C3, C4 and C8 Systems	**GPIB 4-488 $395**
GPIB Software for **OS-65D** (Add -8 for 8" or -5 for 5")	**GPIB 488-D $ 70**
GPIB Software for **OS-65U**	**GPIB 488-U $100**
GPIB Software on two 2716 EPROMS for ROM based systems	**GPIB 488-R $100**
Add Optional **Parallel Printer Interface** to GPIB 4-488	**-P $120**
Add Optional **Calendar/Clock** to GPIB 4-488	**-T $ 25**
Add **2K RAM** to GPIB 4-488 (Specify location, $4000-$BFFF & $D000-$EFFF available)	**-M $25**
GPIB Controller for C1P, Includes Software, Clock, All Features of ROMTERMS, & space for 6K EPROM	**GPIB 6-488R $395**
Add Optional Parallel Printer Interface to GPIB 6-488R	**-P $120**

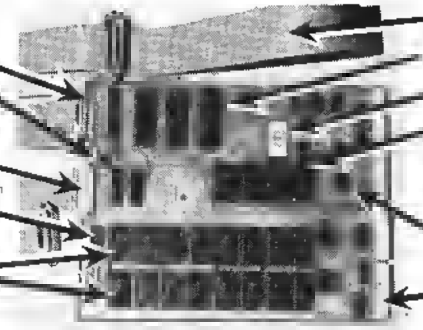
EPROMS:

C1P ROM with 24/48 Col Display for Series II, Smart Terminal, Line Editing, Corrected Keyboard Screen Clear and More	**ROM-TERM II $59.95**
C1P ROM with 24 Col Display. Other ROM-TERM II Features, Disk Boot, and ROM/Disc Basic Interchange	**ROM-TERM $59.95**
C4P-MF/C8P-DF Disk warm start, changed IRQ Vector and just flip switch for Serial or Video System with Corrected Keyboard	**SYNKEY $39.95**

ENHANCED MONITOR ROOMS FOR USE ON GPIB 4-488 & MEM BOARDS:

Expanded Support for C4P & C8P Featuring Calendar/Clock, Line Edit, Smart Terminal, Memory Files, Parallel Printer Control, Corrected Keyboard, All Features of ROMTERMS, **Disk Support** with Warm Start and More	**MI48P1 $59.95**
Expanded C1 Monitor with Calendar/Clock Software, Hard Disk Boot, Warm Start and Multi-User Control for C2 Systems	**MIC2-1 $59.95**

IEEE-488 CONTROLLER INTERFACE



THE GPIB 4-448 INTERFACE BOARD CONVERTS ANY OSI COMPUTER INTO AN IEEE-488 INSTRUMENT BUS CONTROLLER!

BENEFITS — Provides a Sophisticated Instrumentation Controller at very low cost (often saving thousands of Dollars). The combination of IEEE-488 Instrumentation Controller and High Capacity Hard Disk file storage available on OSI Computer systems is available at a fraction of the cost required by the nearest competitor. The IEEE-488 Bus, also known as the GPIB,HP-IB or IEC-625 is the most popular International Standard for connecting instrumentation systems. This 16-line bus is designed to interconnect and control up to 15 instruments at a time. Currently, over 2000 different instruments are available to work on this bus. They include: Plotters, Digitizers, Printers, Graphic Displays, Recorders and a multitude of specialized Test/Measurement Control Equipment.

EPROM-ABLE — Can be used with a C4-P to create a dedicated IEEE-488 controller.

C2-D MULTIPLE USER SYSTEMS

SAVE — 2 and 3 user Time Sharing Systems are available on the C2-D Winchester Disk Computer at a considerable cost savings from C3 Multiple User Systems. The 3 user C2-D System can be expanded to include a word processing printer, 4 other parallel printers and 3 serial printer interfaces.

COMPATABLE — The special C2-D Multi-User Executive Program is 100% compatable with OS-65U V1.2. The Multi-User Real Time Clock, Memory Partition Control and IRQ Interrupt Management are done on the Micro Interface Memory Board. Thus, the CPU board is not modified and remains in factory condition.

CONVERSIONS — The Up-Grade of your existing C2-D Computer to Multiple User Configuration is also available. Call for details.

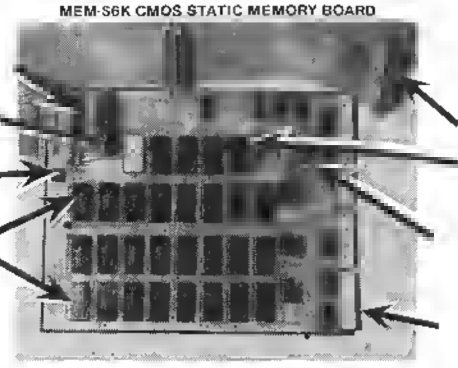
FLOPPY DISK UPGRADES FOR C1P, C4P & C8P

Our Memory/Floppy Board provides easy conversion of 502 and 600 CPU Computers to Floppy Disk Operation. The **MEMF** Board has a floppy disk interface which includes a data separator and the ability to automatically lift the disk drive heads — your floppy disk lifetime will be extended many times. You will retain the cassette interface for your existing software; which can easily be converted to Disk.

This **MEMF-16K** Board is populated with **16K RAM** (50K possible) and has features of the MEM CMOS Static Memory Board with an added floppy interface. The low power memory means extra power supply not required. **ROM Basic** is retained even when Board is populated for 48K Disc Basic. An optional Parallel Printer Port and Real Time Calendar/Clock is on board.

Complete Ready to Run conversion kits with 5¼" or 8" Disk Drives are available

MEM-56K CMOS STATIC MEMORY BOARD



ULTRA-LOW POWER — By using CMOS Static RAM Memory, the total power consumption is about ½ Amp at 5 Volts when populated for 48K. In fact, most of power is used by the Address Line Buffers and the Data Transceivers.

MULTI-USER — Can be addressed for any of the 16 multi-user memory partitions. The low power and single memory board/partition simplify installation and provide a typical $1400 saving for a 3-user system.

MICRO-INTERFACE
3111 SO. VALLEY VIEW BLVD., SUITE I-101
LAS VEGAS, NEVADA 89102
Telephone: (702) 871-3263

Check with your local Dealer or Order Direct.
Phone orders accepted.
TERMS: Check/Money Order/Master Charge/VISA
Sent POSTPAID ON PREPAID ORDERS.
Foreign Orders: Prepaid only.
Add 5% for handling/shipping.

General Purpose Tape I/O for OSI

by Jerry D. Boucher

This relocatable program provides extremely flexible cassette LOAD end SAVE functions. Nearly every locetlon end format can be accommodated.

TAPE LOAD
requires:
OSI C2 Series One or Two cassette recorders

The program can be modified easily to work on other OSI machines.

There are numerous occasions when the cassette tape SAVE and LOAD functions on OSI microcomputers are awkward or inadequate. The limitations of 7-bit bytes, language-dependent format, inflexible storage location, and loss of control characters from tape have frequently forced me to write a dedicated I/O routine for each job. A problem occurred when I increased the baud-rate of the tape I/O and needed to copy my library of tapes at a higher speed. Copying the mixed format (machine language and CHECKSUM) of the Assembler/Editor, Extended Monitor and my own program packages was almost impossible with the existing firmware. The general purpose I/O program presented in listing 1 was my solution to that and other serial I/O problems.

My general purpose program will permit the transfer of data from one tape to another, regardless of the format or content of the data. The program can also be used to LOAD non-standard data into the computer's memory for use with other programs. For example, data may be loaded with

Listing 1: **Tape LOAD and SAVE Routine, Assembly Language.**

```
0000                ;TAPE LOAD AND SAVE
0000                ;J.D. BOUCHER,  8/31/81
0000                ;
1000                         *   = $1000
1000                         MEM = $E0
1000 4C1C10         START    JMP 53
1003 A008           ST       LDY #$08      FIRST PROMPT
1005 B9D210         S1       LDA TAB1,Y
1008 202DBF                  JSR $BF2D
100B 88                      DEY
100C D0F7                    BNE 51
100F 2000FD         S2       JSR $FD00
1011 C97F                    CMP #$7F
1013 F0F9                    BEQ S2
1015 202DBF                  JSR $BF2D
1018 C959                    CMP #$59      YES
101A D008                    BNE SP1
101C A911           S3       LDA #$11      SET MEMORY PAGE
101E 85E1                    STA MEM+1
1020 A900                    LDA #$00
1022 85E0                    STA MEM
1024 A007           SP1      LDY #$07      SECOND PROMPT
1026 B9DA10         SP2      LDA TAB2,Y
1029 202DBF                  JSR $BF2D
102C 88                      DEY
102D D0F7                    BNE SP2
102F 2000FD         SP3      JSR $FD00     GET L OR S
1032 202DBF                  JSR $BF2D
1035 48                      PHA
1036 A009                    LDY #$09      THIRD PROMPT
1038 B9E210         TP1      LDA TAB3,Y
103B 202DBF                  JSR $BF2D
103E 88                      DEY
103F D0F7                    BNE TP1
1041 68                      PLA         CHECK L OR S
1042 C94C                    CMP #$4C    LOAD
1044 F006                    BEQ IN
1046 C953                    CMP #$53    SAVE
1048 F03F                    BEQ OUT
104A D0E3                    BNE SP3
104C 2000FD         IN       JSR $FD00     GET START MARK
104F C90D                    CMP #$0D      IGNORE IF RETURN
1051 F01B                    BEQ IN2
1053 8D6510                  STA STMARK+1
1056 202DBF                  JSR $BF2D
1059 20AD10                  JSR SKIP
105C 20C210         IN1      JSR RUBCHK
105F B0A2                    BCS ST
1061 20B810                  JSR INPUT     READ INPUT UNTIL
1064 C92E           STMARK   CMP #$2E      START MARK FOUND
1066 D0F4                    BNE IN1
1068 91E0                    STA (MEM),Y
106A 202DBF                  JSR $BF2D
106D C8                      INY
106E 20AD10         IN2      JSR SKIP
1071 20C210         IN3      JSR RUBCHK
1074 B08D           IN4      BCS ST
```

(Continued)

Listing 1 *(Continued)*

```
580 1076 20B810            JSR INPUT   LOAD AND STORE
590 1079 202DBF            JSR $BF2D   ALL AFTER START
600 107C 91E0              STA (MEM1,Y
610 107E D1E0              CMP (MEM),Y
620 1080 D0B1              BNE ST
630 1082 C8                INY
640 1083 D0EC              BNE 1N3
650 1085 E6E1              INC MEM+1
660 1087 D0E8              BNE 1N3
670 1089 2000FD     OUT    JSR $FD00   NEW START MARK?
680 108C C90D              CMP #$0D    IGNORE IF RETURN.
690 108E F006              BEQ OUT1
700 1090 202DBF            JSR $BF2D
710 1093 2015BF            JSR $BF15
720 1096 20AD10     OUT1   JSR SKIP
730 1099 20C210     OUT2   JSR RUBCHK
740 109C B0D6              BCS IN4
750 109E B1E0              LDA (MEM).Y
760 10A0 2015BF            JSR $BF15
770 10A3 202DBF            JSR $BF2D
780 10A6 C8                INY
790 10A7 D0F0              BNE OUT2
800 10A9 E6E1              INC MEM+1
810 10AB D0EC              BNE OUT2
820 10AD A30A       SKIP   LDA #$0A
830 10AF 202DBF            JSR $BF2D
840 10B2 A90D              LDA #$0D
850 10B4 202DBF            JSR $BF2D
860 10B7 60                RTS
870 10B8 AD00FC     INPUT  LDA $FC00
880 10BB 4A                LSR A
890 10BC 90FA              BCC INPUT
900 10BE AD01FC            LDA $FC01
910 10C1 60                RTS
920 10C2 A940       RUBCHK LDA #$40
930 10C4 8D00DF            STA $DF00
940 10C7 AD00DF            LDA $DF00
950 10CA C904              CMP #$04
960 10CC F002              BEQ RB2
970 10CE 18                CLC
980 10CF 60                RTS
990 10D0 38         RB2    SEC
1000 10D1 60               RTS
1010 10D2 203F      TAB1   .DBYTE $203F,$5A49,$4E49,$0D0A
1010 10D4 5A49
1010 10D6 4E49
1010 10D8 0D0A
1020 10DA 203F      TAB2   .DBYTE $203F.$532F,$4C0D.$0A00
1020 10DC 532F
1020 10DE 4C0D
1020 10E0 0A00
1030 10E2 203F      TAB3   .DBYTE $203F.$5452.$4154.$530D.$0A00
1030 10E4 5452
1030 10E6 4154
1030 10E8 530D
1030 10EA 0A00
```

the program, held in memory, and retrieved with PEEK statements for processing in BASIC. Or straight text may be stored and LOADed on tapes for use with a word processor. The program can be used in conjunction with the Monitor or Extended Monitor to inspect the contents of a tape for format or for bug-hunting. It can also be used to SAVE any portion of the computer's memory to tape; for example, tokenized BASIC programs.

The portion of the program that actually performs the LOADs and SAVEs is quite simple. Any string of characters present at the serial input port is sequentially stored in memory with a LOAD. With a SAVE the string is sequentially routed to the serial output port. This string includes control characters, line-feeds, data, or any valid ASCII character that might be on the tape. The rest of the program, occupying most of the code, makes the LOADs and SAVEs flexibly controllable and the operation convenient.

Location and Machine-Dependent Features

The program utilizes several routines from the Monitor and BASIC ROMs of the OSI C2 series. If you have a different machine, you may need to change the addresses of these routines and ports:

$BF15	serial output routine
$BF2D	CRT display routine
$DF00	scanned keyboard port
$FC00, FC01	serial I/O port
$FD00	keyboard fetch routine

The ROM routines use locations in the first three pages of memory, so storage of the machine-language program must be in page 3 or above. The program, as shown in listing 1, is written to occupy page 16 ($1000 to $10EA), with data storage beginning at page 17 ($1100). Page zero locations $E0, $E1 are used. However, these locations are not affected by running BASIC, so the program can be called as a USR function or loaded with a BASIC routine.

The program can be relocated with the Assembler/Editor, Extended Monitor, or with the BASIC loader presented below. If the program is to be relocated and directly entered into the computer with the ROM Monitor, change all occurrences of byte $10 to the page number (hex) of the new location. You can change the location of data storage by entering the page number of the start of data storage at line 180 of listing 1.

Listing 2 is a BASIC program which will load and locate the machine language program. Upon RUN the program calls for the page (decimal) where the program is to be located. Data storage is set for the next page. This BASIC loader requires the first nine pages for operation, so the lowest page available for the main program is 10. After the loader has placed the main program in memory, control transfers to the main program *via* a USR instruction in line 120 of the BASIC loader.

Operation

When you turn your machine on, the prompt "L/S?" is displayed. Enter L for LOAD or S for SAVE. The prompt "START?" will then be displayed. If a carriage return is entered, the program immediately begins to load and store whatever is coming into the serial input port, or output whatever is in memory — depending upon whether L or S was selected. If any other key is pressed in response to "START?" that character becomes a start mark.

In the LOAD mode the input is monitored until the start mark appears on the tape. The start mark is then stored in the first memory location, and all subsequent data are stored sequentially in the following memory. For example, machine language programs usually begin with a period to set the monitor in the address mode. If a period is entered as a start mark, any characters on the tape preceding a machine language program will be ignored. Likewise, a semi-colon could be used to select a CHECKSUM program, or you may use special characters for file separation.

Listing 2: BASIC Loader and Relocater.

```
10 REM --RELOCATE AND LOAD MACHINE LANGUAGE PROGRAM--
20 REM --J.D. BOUCHER,  8/31/81
30 PRINT 'ENTER PAGE IN DECIMAL':INPUT P
40 IF P<10 THEN PRINT'TOO SMALL':GOTO30
50 X=P*256: POKE 133,255: POKE 134,P-1
60 FOR J=0 TO 239: Y=X+J
70 READ N: IF N=16 THEN N=P
80 IF J=29 THEN N=P+1
90 POKE Y,N
100 NEXTJ
120 POKE 11,0: POKE 12,P:X=USR(X)
1000 REM -- LOAD AND SAVE PROGRAM--
1001 DATA  76,  28,  16, 160,   8, 185, 210,  16,  32,  45
1002 DATA 191, 136, 208, 247,  32,   0, 253, 201, 127, 240
1003 DATA 249,  32,  45, 191, 201,  89, 208,   8, 169,  17
1004 DATA 133, 225, 169,   0, 133, 224, 160,   7, 185, 218
1005 DATA  16,  32,  45, 191, 136, 208, 247,  32,   0, 253
1006 DATA  32,  45, 191,  72, 160,   9, 185, 226,  16,  32
1007 DATA  45, 191, 136, 208, 247, 104, 201,  76, 240,   6
1008 DATA 201,  83, 240,  63, 208, 227,  32,   0, 253, 201
1009 DATA  13, 240,  27, 141, 101,  16,  32,  45, 191,  32
1010 DATA 173,  16,  32, 194,  16, 176, 162,  32, 184,  16
1011 DATA 201,  46, 208, 244, 145, 224,  32,  45, 191, 200
1012 DATA  32, 173,  16,  32, 194,  16, 176, 141,  32, 184
1013 DATA  16,  32,  45, 191, 145, 224, 209, 224, 208, 129
1014 DATA 200, 208, 236, 238, 225, 208, 232,  32,   0, 253
1015 DATA 201,  13, 240,   6,  32,  45, 191,  32,  21, 191
1016 DATA  32, 173,  16,  32, 194,  16, 176, 214, 177, 224
1017 DATA  32,  21, 191,  32,  45, 191, 200, 208, 240, 230
1018 DATA 225, 208, 236, 169,  10,  32,  45, 191, 169,  13
1019 DATA  32,  45, 191,  96, 173,   0, 252,  74, 144, 250
1020 DATA 173,   1, 252,  96, 169,  64, 141,   0, 223, 173
1021 DATA   0, 223, 201,   4, 240,   2,  24,  96,  56,  96
1022 DATA  32,  63,  90,  73,  78,  73,  13,  10,  32,  63
1023 DATA  83,  47,  76,  13,  10,   0,  32,  63,  84,  82
1024 DATA  65,  84,  83,  13,  10,   0,   0,   0,   0,   0
```

In the SAVE mode, the start mark is not used for control. If any character other than a carriage return is entered as a start mark, that character is output to the tape port before the data are dumped. This adds the start mark to the SAVEd data for future use.

While operating in the SAVE or LOAD mode, the program may be interrupted by depressing the RUB-OUT key. On RUB-OUT, the prompt "INIZ?" appears. If "Y" for YES is entered, the memory will be reset to the beginning, and the L/S prompt will reappear. If any other key is depressed, memory will not be reset before moving to the L/S call. This function allows multiple data sets to be LOADed. After LOAD, the memory must be initialized before SAVE.

Neither SAVE nor LOAD has a termination point. The program will continue to LOAD or SAVE data until RUB-OUT or BREAK is entered. However, there is an echo-check at line 610 in listing 1. This will send control to the "INIZ?" point if the available RAM is exhausted. The program, as written, has no provision for return from a BASIC USR call. Adding the following lines to listing 1 will return control to BASIC if "R" is pressed at "INIZ?":

```
161  BEQ S3
162  CMP #$52  R FOR RETURN
171  RTS       RETURN TO BASIC
```

This package has become a very useful addition to my program library. If you have difficulty getting things in and out of your machine you should give it a try.

Dr. Jerry D. Boucher is a Research Associate at the East-West Center in Honolulu, Hawaii, specializing in cross-cultural psychological problems. He uses his OSI C2-4P for statistical analysis, content-analysis of language, and text processing. Contact Dr. Boucher at East-West Center, 1777 East-West Rd., Honolulu, HI 96848.

MICRO

A Real Tape Operating System

by Dale De Priest

The Commodore PET offers one of the most reliable cassette operating systems. This article describes how it works and offers tips on how to get the most in convenience and reliability from the system.

When I was shopping for a home computer, one of my selection criteria was that the machine not require expensive add-on items, such as a disk drive, before I could use it. Therefore, I needed a suitable cassette system. Unfortunately most cassette systems are either unreliable or very difficult to operate. I decided on the Commodore PET because of its excellent cassette system. However, there are a few tricks to getting the most from the PET's cassette system.

So what makes the PET's system different? First, Commodore modified the standard audio cassette recorder especially for computer use. No tricky adjustment of volume is necessary to read (play) your programs into the machine. Although there is remote control of the cassette drive motor, Commodore's software allows you to regain control of the cassette for manual operation. Finally, an added switch can tell the computer when one of the motion control buttons has been depressed. We will explore each one of these features in detail.

Dual Cassettes

A disk drive can read a block of data into the machine and then rewrite data out on disk. The PET provides two cassette interfaces for this kind of operation. One interface can be used in the read (play) mode to read in the old data. A second cassette can be placed in the write (record) mode to receive the new data. The computer can have complete control of this operation.

You could also use the second cassette to save a back-up copy of your program at the same time you save the original, because the PET always sends the data to both cassette interfaces. Unfortunately, if your second drive came from Commodore you can't use this feature; Commodore designed the drive to shut off in the middle of the recording. If you are using another recorder without remote hook-up (a configuration not recommended by Commodore), simply set the second machine to record before you start the first. [*Editor's note:* A method of modifying standard (less expensive) cassette recorders for use with the PET was described by Jerry Froelich, MICRO 34:81.]

Header Records

Commodore supports two types of files on the tape. You can store programs on your tape with a special header record that contains the name of your program, or you can store data that will also have a special header record. In addition, data files can be several records long. All header records and all data records are buffered in a special place in memory. Each cassette interface occupies a block of memory 192 bytes in length. The first byte of this buffer contains a code that lets the PET know what kind of record it is processing. The other 191 bytes are the record's actual data. The coding for this byte follows:

1. program header
2. data file
3. not used
4. data header
5. end of tape mark

Note that although program storage itself does not use this buffer, the header record containing the name of the program *does* use the buffer. The program name begins in byte 6 of the header and extends for at least 128 bytes, if needed. Searches (and the resulting displays) will only act upon the first 16 bytes. When I save a program I normally save the date out beyond the 16th position. Bytes 2 and 3 contain the start address for program loading. Bytes 3 and 4 contain the end address +1. The normal SAVE command will always default to a start address of 1024. However, SAVEs done with the machine language monitor can have any address. Upgrade and 4.0 ROMs behave just a little differently in this area. The default save location is contained in the start-of-BASIC text pointer. This pointer is initialized to point to 1025.

Load commands always use the header data. On a load, the PET can distinguish between a program file and a data file even if they both have the same name. The PET will load your program where the header tells it to. The RUN command, however, always starts executing at the location pointed to by the start-of-BASIC text pointer.

If the PET encounters an end-of-tape header while searching, it will stop and display the "file not found" error message. I find this very useful — it avoids running down the whole length of tape when a program is not found. For this reason I always put an "end-of-tape" mark at the logical end of every tape. There are two ways to do this. One way is to add a 2 at the end of your SAVE command. For example, SAVE "NAME",1,2 will add an end-of-tape mark after saving the program "NAME" on tape number one. The second way is to specify that you want an end-of-tape mark with the OPEN command. Either method will work, but I prefer the second. I always end my tapes with the following command executed in the immediate mode.

```
OPEN 1, 1, 2, "END OF TAPE" : CLOSE 1
```

Data Files

You can extend the amount of work that can be done with limited memory through the use of data files. Information that would normally occupy memory space in DATA statements can be kept on tape instead. The PET provides for data operations through OPEN statements followed by INPUT# or GET# statements. OPEN tells the computer what you want to do with the file, where the file is, the name of the

file, and the logical number of the file. This number eliminates the need for future commands to repeat all the foregoing data. When a file is opened to read from tape, the computer immediately searches for the file header and then stops. The PET is now positioned correctly in front of the data and knows that you want to read it in. The next INPUT# command to reference that logical file number will read in the first of the data. This command works exactly like the standard INPUT command and is subject to the same 80-character limitation. The GET# command lets you evade this limitation. Since the operating system provides for multiple records in the same file, there must be a special end-of-file indicator. When the file was originally closed, the PET wrote the last of the data on the tape and rhen added one byte of zero at the end. Since the data is written to tape in ASCII format, tbere shouldn't be a zero byte in the data. This then becomes the marker for the end of file.

The GET# command works just like the standard GET command except that its data comes from the cassette huffer insread of tbe keyboard buffer. Therefore, each of the 191 bytes will be read one byte at a time. Rememher that the 192nd byte was reserved by the system to indicate that this is a data file. All of the carriage returns and the commas that would normally be ignored by the INPUT# command will be read by the GET# command. For this reason you cannot use the GET# command wirh numeric variahles. Always use string variables for this command.

At tbis point I would like to take exception to the recommendation in the PET manual that you put the data first if you want to mix data and programs on the same tape. Doing as the manual advises means having to rewind the tape to read in the data after the program has loaded. I always put the data after the program so the program can find it without my help.

When I update files, I always have the program first save itself, and then the data files. I store only one such program and its associated data files on the tape. Therefore, when I open tbe data file, I set the secondary address so that an end-of-tape header will he written when the file is closed. Note that the SAVE command can bc issued by the program. This will not return you to immediate mode, so your program will continue running. Each time I wish to save new data from a program run, I use a different rape. Actually, I alternate hetween two different tapes — one provides the backup for the other. In this manner, if there is a prohlem with the SAVE, I'll only lose the last update and not the whole file. I would also recommend thar your program keep rrack of the revision level of the updates. This can he done by incrementing a counter stored as the firsr record. Revision information can also he stored as a part of the header record when you save it each time. This can he very important if you forget what your last tape was.

Several data files can be maintained on the same tape.The OPEN statement will search for the proper file by name in the same manner as the LOAD command. The only problem is the time involved. The PET puts about a 13-second gap between files.

Error Checking

Good error-checking is an essential component in the design of an adequate tape storage system. One approach to error-checking is to add a parity bit to each character as it is written to tape. The parity bit works by counting each bit as it is sent to the cassette and making sure that the total for each character is an odd number. When reading this data, a count can also be performed to verify that all the bits are read. Another way to check data is to count the number of bits in the whole block, then write a character representing this sum on the tape at the end of the block. This is called a "checksum." If you know how many characters were written on the tape, another check would be to insure that the same number is read back. How many of these checks are available on the PET? All of them! They are kept in a status word which may be examined at any time. This status word contains the result of the last input or output operation.

The PET takes error detection one step further because it includes error *correction* on the tape files. The PET actually stores two copies of every program on the tape. When the tape is read, it not only checks the parity but keeps track of any places with bad parity. When the second pass is reached, it simply substitutes the good data from the second pass for the bad data picked up on the first pass. The PET keeps track of up to 32 bad characters in each record. If this number is exceeded, the load results in an uncorrectable read error. You can check the number of read errors by dividing the number in location 630 by 2 (192 on upgrade and 4.0 PETS). The next location in memory repeats this for the second cassette.

The uncorrectable read error status bit is the only one that will cause a load error. If two data bits are bad in the same character, the correction circuit will miss it. A quick check of ST reveals that the checksum will usually catch this kind of error. For this reason, if you want to be very sure of a good load, insert the following line as the first line of your program:

```
1 IF ST THEN PRINT "ERROR" ST : END
```

You can still run the program by typing in CONT after it stops. A program is stored as one block of data on the tape. Therefore, exceptionally long programs should be avoided if maximum error checking/correcting is desired. Alternately, the monitor can be used to save programs in pieces. The last program piece saved should be the one with the highest address.

Even with all of these schemes we still are not guaranteed to have a good tape. The PET provides a way to make sure the tape can be read by using the VERIFY command. This command reads each character from the tape and compares it with the one in memory. This should insure that the tape was written correctly. Therefore you should always verify a program after you save it.

The VERIFY command can also be used to position the tape. Whenever this command is issued it starts the tape and reads in the next program (or the one specified). The comparison is made without disturbing whatever you have in core. The tape is left positioned just beyond the program — exactly the goal we were trying to achieve. This positioning capability allows us to add programs at the end of the tape, or to modify an existing program and rewrite it in the same place. The PET puts a long leader in between each program so that even if your new version is a little longer, it will still fit. But be careful!

The VIC uses a cassette system that is almost identical to the standard PET. There are only three major differences. First, the VIC only supports one cassette; address 2 is used to support the RS-232 interface. Second, the VIC contains a relocating loader that will automatically start the tape load at the location designated by the start-of-BASIC text pointer. The VIC uses the start and end addresses in the header to calculate the length of the program which is then added to the start-of-BASIC text pointer to arrive at a new end address. The third change is related to the second. Since you might not want a program to be relocated, a new header type has been created. If the first byte of the header contains a three, then the load will work by using the start and end pointers exactly from the program header. This is required when loading most machine language programs.

Please send any comments or questions to the author at the following address: 611 Galen Drive, San Jose, CA 95123.

MICRO

COPCOP Single Drive Copier

by Peter Kleijnjan

COPCOP is a versatile disk copying program for OSI C2-4P and C4P-MF systems. With it, the user can copy diskettes quickly and easily, regardless of the particular system's configuration.

COPCOP
requires:
OSI with one disk drive

This utility for OSI C2-4P and C4P-MF users features a command file to combine the comfort of a BASIC dialogue with the efficient memory use of machine code.

COPCOP adapts to the size of your system's memory. On a 24K system it copies up to 8 tracks at a time; with 48K, 20 can be accommodated. It also maintains the original sector layout, and automatically skips tracks without headers. COPCOP can print out a neat sector directory (and replace SECDIR). If required, it can copy track 0 or initialize before a write. And of course it's selective, allowing you to specify which tracks to copy.

A vital mechanism used in this program is the so-called "command file." OS-65D can get its input from a variety of input devices: device #1 is the serial terminal, #2 is the keyboard console, etc. What interests us is device #5 — use of memory as input. This means that you can POKE into memory *any* sequence of inputs to OS-65D. (Consequently, you can input to BASIC, EM, or even assembler, since they all use OS-65D routines.) Control can then be transferred to memory by giving the "IO 10,02" command.

COPCOP first collects the data it needs to copy a disk, and then builds a command file using these data. This command file resides from \$4800 upwards (figure 1). It uses the "getkey" routine at \$252B to halt execution when you have to exchange diskettes.

What is the advantage of this technique over simple DOS commands embedded (with DISK!" ") in the BASIC program? After control has been transferred to the command file, the BASIC program itself isn't needed anymore! Nor is the 8K BASIC interpreter. This saves enough memory (about 10K) to enable copying of four more tracks at a time.

Listing 1 is the program itself. Lines 100-273 extract necessary information from both the operator and the source disk (the "original"). The latter is achieved by use of the "DI xx" DOS command, which prints a sector map of track xx. Lines 500-590 are a subroutine that prints a sector directory for all requested tracks.

Lines 1000-1280 contain the diskette copy subroutine. This routine calls 2000-2130, which adds a single pass (one series of CAlls and SAves) to the command file. 5000-5130 prints an appropriate CAll or SAve command to the command file, and 7000-7060 converts a page number to a full hexadecimal address.

Please address correspondence to Kleijnjan Consultants BV, Kerkwetering 11, 3421 TS Oudewater, The Netherlands.

Listing 1

```
1 REM COPCOP REL4.0 24-48K, TR.0, IN, RUNS ALSO ON V3.2MOD
5 POKE133,71:RUN10
10 FORI=1TO20:PRINT:NEXT
20 PRINT"**** COPY COPY ****":FORI=1TO7:PRINT:NEXT
30 PRINT"----------------------------------------"
40 PRINT"Kleijnjan Consultants              (c) 1980"
50 PRINT"----------------------------------------":PRINT
60 PRINT"This program contains a FORMAT LISTER and a DISKETTE COPIER."
70 PRINT"The copier has facilities to copy track zero and to";
75 PRINT" initialize":PRINT"before write."
80 PRINT
100 POKE2893,28:POKE2E94,11:POKE2888,0:POKE8722,0
110 DIMD$(39,E):CR$=CHR$(13):MA=INT((PEEK(8960)-29)/E)
111 X$=""
112 PRINT:PRINT"Which "+X$+"tracks should NOT be copied or listed?"
113 INPUT"(XX-YY)          (RETURN to continue)";K$:PRINT
115 IFK$=""THEN120
116 IFLEN(K$)<>5THENPRINT:PRINT"--LENGTH ERROR--":GOTO112
117 FT=VAL(LEFT$(K$,2)):LT=VAL(RIGHT$(K$,2))
118 FORI=FTTOLT:D$(I,1)="N":NEXT
119 X$="other ":GOTO112
120 PRINT:PRINT"Insert the disk you want copied, then hit any key."
121 I=0:PRINT:DISK!"GO 252B":PRINT
122 I=I+1
123 IFI=40THEN280
124 IFD$(I,1)="N"THEN122
130 I$=STR$(I)
140 I$=RIGHT$("0"+RIGHT$(I$,LEN(I$)-1),2)
160 DISK!"ME D100,D100
170 DISK!"IO ,10
1E0 DISK!"DI "+I$
190 DISK!"IO ,02
200 PRINT#5:PRINT#5,"*"
210 DISK!"ME D100,D100
220 INPUT#5,A$
230 IFLEFT$(A$,3)="TRA"ORA$=""THEN220
240 IFA$="*"THEN270
```

(Continued on next page)

Listing 1 *(Continued)*

```
250 SN=VAL(MID$(A$,2,2))
260 D$(I,SN)=RIGHT$(A$,1):GOTO220
270 DISK!"ME D100,D100"
271 X$="  ":FORK=1TO5:X$=X$+X$:NEXT
272 PRINT#5,X$;:PRINT#9
273 GOTO122
280 PRINT:INPUT"Format listing";X$:PRINT
290 IFLEFT$(X$,1)="Y"ORLEFT$(X$,1)="J"THENGOSUB500
300 PRINT:INPUT"Copy diskette";X$:PRINT
310 IFLEFT$(X$,1)<>"N"THENX$="":GOSUB1000
320 END
500 PRINT:INPUT"List on printer";X$
501 IFLEFT$(X$,1)<>"Y"THEND=2:GOTO503
502 D=1
503 FORJ=1TO39
505 IFD$(J,1)="N"THEN5E0
510 PRINT#D,"Track";J;
520 IFD$(J,1)=""THENPRINT#D,TAB(12);"Missing header":GOTO580
530 K=0
540 K=K+1
550 IFD$(J,K)=""ORK=9THEN5E0
560 PRINT#D,TAB(12);"sector";K;": ";D$(J,K);" page(s)"
570 GOTO540
5E0 NEXTJ
590 PRINT:PRINT:RETURN
998 DATA2,10,18,26,56,64,80,88,96,104,112,120,128,136,144
999 DATA152,1E0,16E,17E,184
1000 REM--COPY SUBROUTINE
1080 DISK!"ME F000,4E00":PRINT#5,"EXIT";CR$;:PRINT#9
1090 TF=1:PRINT:INPUT"Initialize before write (Y/N)";IX$:PRINT
1092 PRINT" -just a few seconds.":PRINT
1095 GOSUB2000
1096 TL=I-1:IFCN=0THEN1150
1097 PRINT#5,CR$;"Place original";CR$;:PRINT#9
109E PRINT#5,"GO 252B";CR$;:PRINT#9
1100 X$="CALL":GOSUB5000
1105 PRINT#5,CR$;"Place copy";CR$;:PRINT#9
1110 PRINT#5,"GO 252B";CR$;:PRINT#9
1120 X$="SAVE":GOSUB5000
1135 TF=TL+1
1140 IFTF<40THEN1095
1150 INPUT"Track zero copy";K$:IFLEFT$(K$,1)<>"Y"THEN1240
11E0 PRINT#5,CR$;"Place systemdisk";CR$;"GO 252B";CR$;
1164 PRINT#5,"CA 0200=13,1";CR$;CR$;"Place original";CR$;
11E8 PRINT#5,"GO 252B";CR$;"GO 0200";CR$;"2";CR$;"R4000";CR$;
1170 PRINT#5,"E";CR$;CR$;"Place copy";CR$;:PRINT#9
1174 PRINT#5,"GO 252B";CR$;"GO 0200";CR$;"2";CR$;
117E PRINT#5,"W4000/2200,8";CR$;"E";CR$;:PRINT#9
1240 POKE10944,7E:POKE10945,81:POKE10946,42
1250 PRINT#5,"GO FFA0"
1260 PRINT:PRINT"Press any key to start and to continue."
12E0 DISK!"ME 4800,F000":DISK!"IO 10,02":RETURN
2000 I=TF:CN=0
2005 IFI>39THEN2120
2010 IFD$(1,1)=""ORD$(I,1)="N"THENI=I+1:GOTO2005
2020 READFV
2030 K=1
2040 GOSUB7000:REM CONVERT FV TO F$: 10 BECOMES 0A00
2050 IFD$(I,K)=""ORK=9THEN2100
2060 FV=FV+VAL(D$(I,K))
2070 D$(I,K)=F$+"/"+D$(I,K)
2080 K=K+1
2090 GOTO2040
2100 I=I+1:CN=CN+1
2110 IFCN<MATHEN2005
2120 RESTORE
2130 RETURN
5000 FORJ=TFTOTL
5010 K=1
5020 IFD$(J,K)=""ORD$(J,K)="N"THEN5120
5030 C$=LEFT$(D$(J,K),4)
5040 J$=STR$(J):J$=RIGHT$("0"+RIGHT$(J$,LEN(J$)-1),2)
5050 K$=RIGHT$(STR$(K),1)
50E0 IFX$="CALL"THEN5085
5070 IFIX$<>"Y"ORK<>1THEN5080
5075 PRINT#5,"IN "+J$;CR$;:PRINT#9
5080 PRINT#5,"SA "+J$+","+K$+"="+D$(J,K);CR$;:PRINT#9:GOTO5090
5085 PRINT#5,"CA "+C$+"="+J$+","+K$;CR$;:PRINT#9
5090 K=K+1
5110 IFK<MATHEN5020
5120 NEXT:RETURN
7000 X=INT(FV/16)+48
7010 IFX>57THENX=X+7
7020 LD$=CHR$(X)
7030 X=FV-16*INT(FV/16)+48
7040 IFX>57THENX=X+7
7050 F$=LD$+CHR$(X)+"00":RETURN
```

EXIT
02 TRACK.
A*__
A*Place original
A*GO 252B
A*CA 0200=12,1
A*CA 0300=12,2
...
...
A*CA 3000=19,1
A*__
A*Place copy
A*GO 252B
A*SA 12,1=0200/1
A*SA 12,2=0300/1
...
...
A*SA 19,1=3000/8
A*__
A*Place original
(etc.)

***Figure 1:* OS-65D command file: input from memory (from the actual command file) is underlined. The instructions ("Place copy") are only for the benefit of the operator.**

"YOUR SALVATION IN THE SEA OF INFLATION"

ARK COMPUTING

80 Column Card & 16K RAM Board Sale

The best price available on a 80-Column for your APPLE.
Wizard-80 by Wesper-Micro
RP $345.00 ARK $225.00

16K Ram Board ARK $ 89.00
32K Ram Board ARK $139.95
- with 16-K ram installed -
(add $ 20.00 for full 32-K ram installed)

LOWER CASE PLUS II by Lazer Microsystems.
The Lazer MicroSystems LCP II is the LCA value for the budget minded APPLE II owner. Works with all Revision 7 and Later APPLE II's. Includes Basic and Pascal software on disk. Works with many popular word processors.

ARK's LOW PRICE of $ 19.95

For people who demand the best

The best Lower Case Adapter available for the APPLE II.
LOWER CASE PLUS by Lazer MicroSystems.
This feature packed board has twice the features of competing boards.
The only LCA that works with VISICALC and is recommended by Stoneware for DB MASTER.
FEATURES:
- -Expandable to 4 character sets (2 on board)
- -Inverse only mode
- -Includes graphics & ASCII
- -Compatible w/most word processors available.

AND WITH ARK's LOW PRICE THERE'S NO REASON TO PURCHASE ANY OTHER.

RP $ 64.95 ARK $44.95

Remarkable value in Keyboard Enhancers.
The Lazer MicroSystems Keyboard +Plus with these features:
- -64 character type ahead buffer
- -Buffer can be cleared & disabled
- -Works on all APPLE II's
- -Lets you use the shift-key as a typewriter shift-key, allowing you to enter directly the 128 ASCII character set from the APPLE keyboard.

KEYBOARD +PLUS
RP$ 99.95 ARK $ 69.95

THREE EXCITING PRODUCTS. A ONCE IN A LIFETIME DEAL. LOWER CASE PLUS+, KEYBOARD PLUS+, & AN 80-COLUMN BOARD ALL FOR LESS THAN THE NORMAL COST OF THE 80-COLUMN BOARD ALONE.

ARK $299.95

software

By LAZER

- -ANIX 1.0: A set of incredible disk utilities w/ UNIX like Operating System. ARK $ 34.95
- -LAZER PASCAL: A unique systems programming language with many features of the 'C' programming language. ARK $ 29.95
- -DISASM/65: Most popular disassembler for the APPLE II. ARK $ 19.95

By DATAMOST

-SNACK ATTACK:	RP$ 29.95	ARK$ 19.95
-THEIF:	RP$ 29.95	ARK$ 19.95
-CONEY ISLAND:	RP$ 29.95	ARK$ 19.95
-6502 ASSEMBLY by RANDY HYDE:	RP$ 19.95	ARK$ 15.00

By BRODERBUND

-MIDNIGHT MAGIC: (Best Pinball game we've seen yet)	RP$ 34.95	ARK$ 24.95
-APPLE PANIC	RP$ 29.95	ARK$ 19.95
-SPACE QUARK	RP$ 29.95	ARK$ 19.95
-RED ALERT	RP$ 29.95	ARK$ 19.95
-TRACK ATTACK	RP$ 29.95	ARK$ 19.95
-CENETIC DRIFT	RP$ 29.95	ARK$ 19.95
-SPACE WARRIOR	RP$ 24.95	ARK$ 17.95

By ON-LINE SYSTEMS

-MOUSKATTACK	RP$ 34.95	ARK$ 24.95
-CROSS FIRE	RP$ 29.95	ARK$ 21.95
-CRANSTON MANOR	RP$ 34.95	ARK$ 24.95
-ULYESSES	RP$ 34.95	ARK$ 24.95
-SOFTPORN ADV	RP$ 29.95	ARK$ 21.95
-TIME ZONE	RP$ 99.95	ARK$ 75.95
-LISA 2.5	RP$ 79.95	ARK$ 59.95
-SPEED/ASM	RP$ 39.95	ARK$ 29.95
-THRESHOLD	RP$ 39.95	ARK$ 29.95
-PEGASUS II	RP$ 29.95	ARK$ 21.95

We accept: MASTERCARD, VISA(Include card # and expiration date), CASHIER or CERTIFIED CHECKS, MONEY ORDERS, or PERSONAL CHECKS(Please allow 10 days to clear).
Please add 3% for shipping & handling (minimum $2.00). Foreign orders please add 10% for shipping & handling.
We accept COD's (Please include $2.00 COD charge)
California residents add 6% sales tax. All equipment is subject to price change and availability without notice. All equipment is new & complete with manufacturer's warranty.

Call us at (714) 735-2250

ARK COMPUTING • P.O. Box 2025 • Corona, CA 91720

Votrax Interface for SYM

by John Valente

Interfece the Votrax Speech Synthesizer to your system's VIA. Although intended for e SYM-1, the techniques described are reedily edeptable to other systems using e 6522.

VOTRAX DRIVER
requires:
SYM-1
Sweet Talker Votrax Interface Board

It is adaptable to other systems (such as AIM) with 6522 VIA.

The Votrax SC-01 Speech Synthesizer IC lets you experiment with computer-generated speech at a reasonahle cost. The Sweet Talker hoard, which includes the Votrax IC, allows easy interfacing to most computers. It is available from The Micro Mint, Inc., 917 Midway, Woodmere, NY 11598. While I will he describing the interface of the Votrax to my SYM-1 6522 VIA, the programs can be readily adapted to other systems using the 6522. This article provides a machine language driver, followed hy a BASIC program to convert the mnemonics for each phoneme of speech into the numerical codes needed by Votrax. (A phoneme is one of the smallest units of speech that distinguishes one word from another; i.e., the *m* in mat and *b* in bat.)

Communication with the Votrax IC resembles a parallel printer interface with handshaking. Figure 1 shows the connections between the 6522 and the Sweet Talker hoard. This example uses VIA #2, port A, accessed through the SYM's "AA" connector. Any other VIA port can be used as long as all eight data bits and the two control lines are available. Be sure the Sweet Talker is connected to your power supply.

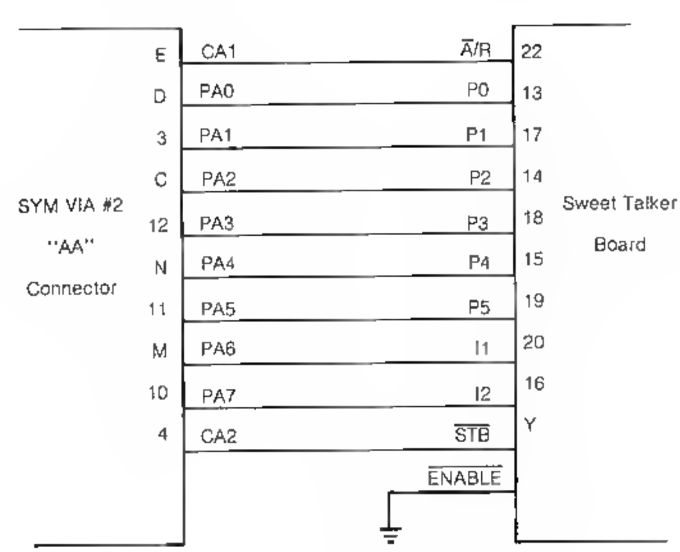
Bits zero through five form the code which tells Votrax which phoneme to synthesize. Bits six and seven select one of four pitch levels for the voice. Control line CA2 latches the data into the Votrax IC and starts synthesis. After the phoneme has sounded, $\overline{A}$/R goes high. The CA1 line on the 6522 senses this transition and generates an interrupt, causing the next phoneme code to be sent out. This process repeats until a "stop" code ($3F) is sent. The $\overline{\text{ENABLE}}$ line is not used and is grounded to allow the board to operate.

Listing 1 sends a sequence of phoneme codes to the Sweet Talker with the proper timing of control signals. It is written in RAE – 1 format (SYM's Resident Assembler and Editor). The sequence of phonemes is limited to 256, due to the eight-bit length of the X index register. A block of memory to store the values for PITCH and the sequence of phonemes starting at PHONEME is reserved following the program. If you relocate this routine, you need only change the references to these locations. If your 6522 is at a different location, the register addresses will have to be modified accordingly.

The program begins by pointing the IRQ vector to the INT routine. The SYM requires a JSR ACCESS before this operation. Then, interrupts are temporarily disabled and the 6522 Interrupt Enable Register (IER) is set up to generate an IRQ on the rising edge of CA1. The port is then configured for output.

You must send the first phoneme code manually. The X register, which is used to index into the PHONEME list, is set to zero. The next few lines of code are needed because of a timing restraint in the Votrax chip. After sending data to Votrax, wait at least 450 nsec hefore sending a strobe pulse to latch it. Since the 6522 normally waits only 300 nsec before pulsing, the handshake/strobe is first inhibited by sending a $01 to the Peripheral Control Register (PCR). The phoneme data is fetched from the list, OR'ed with the value of PITCH to select the voice, and sent down the line. Now the handshake is enabled by sending a $0B to the PCR. A dummy read of the port outputs the pulse, and hy now much more than the required 450 nsec has elapsed.

Figure 1



After the phoneme has sounded, the CA1 line goes high and forces an interrupt. In the INT routine, the same manipulation of the handshake occurs to insure proper timing. The X register is incremented to point to the next phoneme code, and tbe data is fetched. But we don't want to OR the code with the value of PITCH if the "stop" code is encountered. After checking for this, the data and handshake are sent out as before. Conveniently, the dummy read at 1060 leaves the latest phoneme code in the accumulator.

As the program exits the INT routine, it loops continuously to IDLE until the "stop" code is found. Then the IER is altered to ignore further interrupts and the IRQ vector is restored to its original value. The address given is for tbe SYM's Supermon 1.1. Finally, control is restored to the calling program.

You can use this program directly by entering phoneme codes through the monitor, starting at address PHONEME and ending with $3F to end the speech. A value of $00, $40, $80 or $C0 must be entered at location PITCH. Try using the random values in memory as a phoneme list; you will hear some very bizarre sounds.

To compose intelligible words, however, it is more convenient to use mnemonic codes for each phoneme because these are closer to English. Listing 2, a BASIC program, prompts for the standard Votrax mnemonics, translates them into the proper numerical code, then places them sequentially into the PHONEME block. A call to the machine language driver produces the speech. Rememher to reserve space for the machine language routine plus 257 bytes before entering BASIC.

The program is written in Syncrtek Bas-1, but is easily translated into otber BASICs. The &"xxxx" is Bas-1's convention for hexadecimal numbers, and X = USR (address,0) calls a machine language routine. Lines 10-40 list the standard Votrax mnemonics, and lines 130-150 READ them into an array for later comparison. Lines 105-115 assign the machine language addresses to variables; simply cbange these lines to conform to any relocation you have made. Lines 500-540 prompt for the desired phoneme, search for a match, and then POKE the corresponding code into the PHONEME list. When you enter STOP in answer to the prompt, the program asks you to select a pitch

***Listing 1*: Votrax Driver — Assembly Lenguage Listing**

```
                 ;            VOTRAX DRIVER
                 ;
                 ;            by John Valente   Box 9  Marlboro VT 05344
                 ;            Written September 1981
                 ;
                 ;       The following addresses are for VIA #2 on the SYM-1.
                 ;       Change as required for your system.
                 ;
                 PAD         .DE $A801     ;Port A, VIA #2
                 PADD        .DE $A803     ;Port A data direction
                 PCR         .DE $A80C     ;Peripheral Control Register
                 IER         .DE $A80E     ;Interrupt Enable Register
                 ;
                 ;       The following are SYM Supermon 1.1 references:
                 ;
                 ACCESS      .DE $8B86     ;Needed to change vectors
                 IRQVEC      .DE $A67E     ;IRQ vector
                 IRQORIG     .DE $800F     ;Original IRQ service routine
                             .BA $1000     ;Program origin
                             .OS           ;Save object code
                 ;
                 ;       Program starts here:
                 ;
1000- 20 86 8B   START       JSR ACCESS    ;Change IRQVEC: point to INT
1003- A9 48                  LDA #L,INT
1005- 8D 7E A6               STA IRQVEC
1008- A9 10                  LDA #H,INT
100A- 8D 7F A6               STA IRQVEC+1
100D- 78                     SEI           ;Prevent interrupts for now
100E- A9 82                  LDA #%10000010 Enable interrupts on CA1 ^ edge
1010- 8D 0E A8               STA IER
1013- A9 FF                  LDA #$FF      ;Set VIA port A for output
1015- 8D 03 A8               STA PADD
1018- A2 00      FIRST       LDX #$00      ;Initialize phoneme list pointer
101A- A9 01                  LDA #%00000001 Disable handshake/strobe
101C- 8D 0C A8               STA PCR
101F- BD 65 10               LDA PHONEME,X Get first phoneme code
1022- 0D 64 10               ORA PITCH     ;Determine pitch
1025- 8D 01 A8               STA PAD       ;Send code to Votrax
1028- A9 0B                  LDA #%00001011 Now enable handshake/strobe
102A- 8D 0C A8               STA PCR
102D- AD 01 A8               LDA PAD       ;Dummy read: force handshake/strobe
1030- 58                     CLI           ;Now allow interrupts
1031- C9 3F      IDLE        CMP #$3F      ;Found STOP code yet ?
1033- F0 03                  BEQ RETURN    ;Yes, exit
1035- B8                     CLV           ;No, loop until found
1036- 50 F9                  BVC IDLE
1038- A9 02      RETURN      LDA #%00000010 Disable VIA interrupt
103A- 8D 0E A8               STA IER
103D- A9 0F                  LDA #L,IRQORIG Restore original IRQ vector
103F- 8D 7E A6               STA IRQVEC
1042- A9 A6                  LDA #H,IRQVEC
1044- 8D 7F A6               STA IRQVEC+1
1047- 60                     RTS           ;Return to calling program
                 ;
                 ;        Interrupt Service Routine follows:
                 ;
1048- A9 01      INT         LDA #%00000001 Disable handshake as before
104A- 8D 0C A8               STA PCR
104D- E8                     INX           ;Increment pointer to phoneme list
104E- BD 65 10               LDA PHONEME,X Get next phoneme code
1051- C9 3F                  CMP #$3F      ;Is it the STOP code ?
1053- F0 03                  BEQ NOMASK    ;Yes, leave it alone
1055- 0D 64 10   MASK        ORA PITCH     ;No, set the pitch
1058- 8D 01 A8   NOMASK      STA PAD       ;Send code to Votrax
105B- A9 0B                  LDA #%00001011 Now enable handshake/strobe
105D- 8D 0C A8               STA PCR
1060- AD 01 A8               LDA PAD       ;Send strobe:phoneme code in Accum.
1063- 40         DONE        RTI           ;Go back and wait
                 ;
1064-            PITCH       .DS 1         ;Reserve a space for pitch value
1065-            PHONEME     .DS 256       ;Reserve a page for phoneme codes
                             .EN
```

Listing 2: Votrax Phoneme Translator — BASIC Listing and Sample Run

```
1 REM VOTRAX PHONEME TRANSLATOR
2 REM WRITTEN BY JOHN VALENTE BOX 9 MARLBORO VT 05344
3 REM SEPTEMBER 198I
10 DATA EH3,EH2,EH1,PA0,DT,A2,A1,ZH,AH2,I3,I2,II,M,N,B,V
20 DATA CH,SH,Z,AW1,NG,AH1,OO1,OO,L,K,J,H,G,F,D,S
30 DATA A,AY,Y1,UH3,AH,P,O.I,U.Y,T,R,E,W,AE,AE1
40 DATA AW2,UH2,UH1,UH,O2,OI,IU,U1,THV,TH,ER,EH,E1,AW,PA1,STOP
I00 DIM T$(63)
105 M=&"1000":REM ADDRESS OF MACHINE LANGUAGE ROUTINE
II0 L=&"1065":REM ADDRESS OF START OF PHONEME LIST
115 V=&"1064":REM ADDRESS OF PITCH VALUE
120 FOR A=0 TO 63
130 READ P$
I40 T$(A)=P$
150 NEXT A
500 Y=0:INPUT "PHONEME ? ";X$
510 IF X$="STOP" THEN 700
520 IF T$(Y)=X$ THEN POKE L,Y:L=L+I:GOTO 500
525 REM SUBSCRIPT OF MATCHED STRING IS CORRECT PHONEME CODE
530 Y=Y+I
535 IF Y>63 THEN PRINT "NOT A VALID PHONEME. TRY AGAIN. ":GOTO 500
540 GOTO 520
700 POKE L,63
710 PRINT "SELECT PITCH OF VOICE: "
712 PRINT "TYPE EITHER 0,64,128 OR I92 (LOWEST TO HIGHEST PITCH)"
714 INPUT P
715 POKE V,P
720 INPUT "TYPE ANY LETTER AND 'RETURN' TO HEAR YOUR WORD.";D$
730 X=USR(M,0)
740 END
OK
RUN
PHONEME ? H
PHONEME ? EHI
PHONEME ? EH3
PHONEME ? LK
NOT A VALID PHONEME. TRY AGAIN.
PHONEME ? L
PHONEME ? O
PHONEME ? STOP
SELECT PITCH OF VOICE:
TYPE EITHER 0,64,128 OR I92 (LOWEST TO HIGHEST PITCH)
? 64
TYPE ANY LETTER AND 'RETURN' TO HEAR YOUR WORD.R
```

for the voice. After responding to line 720, the machine language driver is called and you will hear the result.

I suggest experimenting with single words before assembling long messages. I think you will find that the components of human speech are very complex. Two words which rhyme to our ears are often composed of different series of phonemes. What might seem to be a simple vowel sound is sometimes a series of two or even three different phonemes. Be sure to include pauses between words (two different pause mnemonics are available).

VOTRAX is a trademark of Federal Screw Works, Inc.

John Valente is interested in using the computer to generate sounds and musical structures unavailable in conventional instruments. He has been published in *Electronotes, Newsletter of the Musical Engineering Group.* You can write to Valente at Box 9, Marlboro, VT 05344.

MICRO

6508 — A New 6502 Configuration

by Ralph Tenny

This new configuration of the 6502 will stimulate a number of very innovative designs, including multiprocessing and two-chip designs.

At long last, there is an improved version of the 6502. The Commodore Semiconductor Group (formerly MOS Technology) has produced the 6508 microcomputer without adding any new instructions.

This 40-pin IC is the familiar 6502 (actually, the 6512, which requires an off-chip clock) with 256 bytes of read/write memory, eight I/O pins and a full complement of address and data lines. In addition, the address lines can be tri-stated to facilitate DMA (Direct Memory Access) operations. Thus, with eight I/O lines and AEC (Address Enable Control) to control the address and data lines during DMA operations, a total of nine functions have heen added to the package.

Because of necessary pinout changes, the following functions are no longer availahle: NMI, RDY, SYNC, S.O., Ø2 Out and DBE. Actually, DBE (Data Bus Availahle) is used on the 6512 to furnish compatihility with the 6800, and is replaced with AEC. The 6512 has three V_{ss} pins, two of which are on the 6508. The one remaining pin fills a formerly unused pin, thus giving a full eight pins for the I/O port.

The I/O port is situated at $0000 (Data Direction Register) and $0001 (Output Register). This location for the port has a number of advantages. I/O operations will he faster and have shorter drive routines, since zero page addressing can he used. However, setting the port to input can result in external hardware that enters data directly into memory, with no intervention by the processor. A recent article detailed the following additional possihilities for the 6508:

1. Multi-processor operation with overlapped memory operations.
2. I/O lines used as segment addresses for over I megahyte addressing.
3. I/O lines used as vector inputs for vectored interrupts.
4. I/O lines used to arhitrate interrupt priorities.
5. Internal memory used to operate supervisory programs during DMA.

The following material bas heen adapted from the 6508 data sheet, and is used with permission of Commodore:

Figure 1 shows the 6508 memory map. Note that page 0 and page 1 overlap in the 256 hytes of on-board read/write memory. Also, the zero page area is further depleted hy two addresses used hy the I/O port. Otherwise, the entire 64K of memory space is availahle for typical 6502 uses.

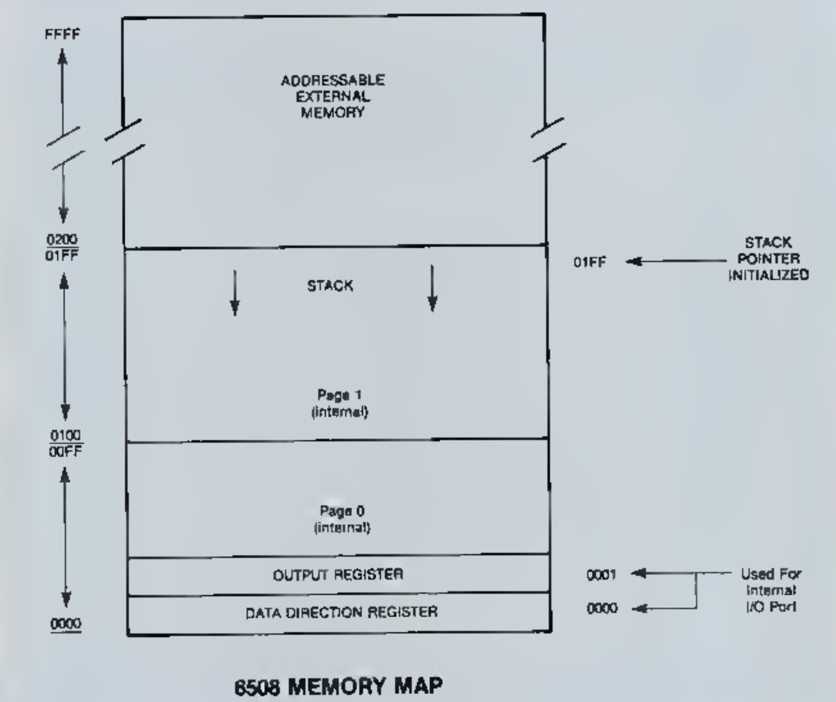


Figure 1: **The memory map for the 6508 differs from that of the 6502 only in the zero page area, and in the addition of dedicated I/O space.**

Figure 2 is a block diagram of the 6508, showing the internal architecture of the processor. This is almost identical to the 6502, except for the obvious addition of I/O port and read/write memory.

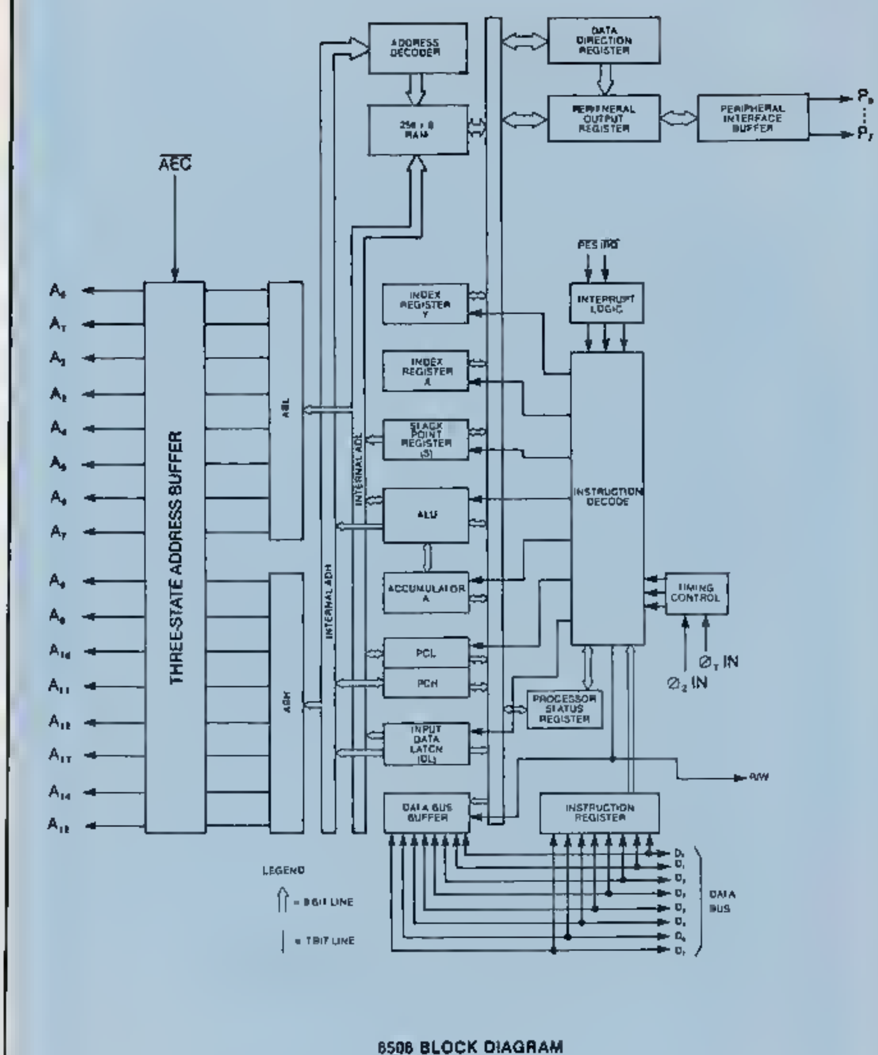


***Figure 2:* Block diagram for the 6508. Obvious additions — Data Direction Register, Peripheral Output Register and associated buffers, and page zero Read/Write memory.**

Signal	Pin		Pin	Signal
RES	1		40	0, IN
0, IN	2		39	R/W
IRQ	3		38	DB_0
AEC	4		37	DB_1
VDD	5		36	DB_2
A_0	6		35	DB_3
A_1	7		34	DB_4
A_2	8		33	DB_5
A_3	9		32	DB_6
A_4	10	6508	31	DB_7
A_5	11		30	P_0
A_6	12		29	P_1
A_7	13		28	P_2
A_8	14		27	P_3
A_9	15		26	P_4
A_{10}	16		25	P_5
A_{11}	17		24	P_6
A_{12}	18		23	P_7
A_{13}	19		22	A_{15}
VSS	20		21	A_{14}

***Figure 3:* New pinout assignments are quite different for the 6508; see text for additional details.**

Figure 3 shows the pinout of the 6508.

I expect the 6508 to be used in innovative designs, both in controller-type applications, and in more sophisticated data communications projects. The greatest advance I see for the controller field is that two-chip designs are possible, if eight or fewer I/O lines are required. In the past, nearly every single-chip processor implementation has required at least three ICs. With the 6508, you need add only an EPROM!

Reference

1. Enhanced CPU's memory, I/O expand its applications; *Electronic Design News*, August 19, 1981, G. Venkatesh, Commodore Semiconductor Group.

Ralph Tenny may be contacted at P.O. Box 545, Richardson, Texas 75080.

MICRO

Time of Day the Easy Way

by Martin De George

If you add the new 6526 Complex Interface Adaptor chip to your computer, you'll have processor-independent real time available. You may also substitute the 6526 for an existing 6522.

The demonstration program included is written for OSI. The hardware is applicable to any 6502 computer.

***Table 1:* Register Functions of 6526**

Register Address	Name	Use
0	PRA	Peripheral Data Register A
1	PRB	Peripheral Data Register B
2	DDRA	Data Direction Register A
3	DDRB	Data Direction Register B
4	TA LO	Timer A LO
5	TA HI	Timer A HI
6	TB LO	Timer B LO
7	TB HI	Timer B HI
8	TOD 10ths	Time of Day Tenths of a Second
9	TOD SEC	Time of Day Seconds
A	TOD MIN	Time of Day Minutes
B	TOD HR	Time of Day Hours
C	SRD	Serial Data Register
D	ICR	Interrupt Control Register
E	CRA	Control Register A
F	CRB	Control Register B

Until now I have been reluctant to implement a real-time clock (RTC) on my system because it was just too much bother, both from software and hardware standpoints. With my OSI system I need a chain of dividers to divide the system clock down and interrupt the system periodically, say every second, so that a routine is executed which updates a counter. It is then necessary to check the counter for rollover at 60, or convert an N-bit number to TOD each time you would like to know the time. Besides this chore, there is the not so trivial problem of ruining a disk access if you allow interrupts during these times. If you disable interrupts during disk access, it is very possible that you will miss an interrupt if you do a lot of disk access. I wanted a TOD chip that was easy to install and access without tying up my system with unnecessary overhead.

Just recently the people at Commodore introduced a gem of a chip called the 6526 Complex Interface Adapter (CIA). Don't let the name scare you; it's only complex in what it can do, not in how to do it.

Basically this device is a sophisticated 6522 like those found on the AIM and many other computers. The one major difference is that it contains a time-of-day clock function which reads out TOD in Hours, Minutes, Seconds, and Tenths of Seconds. It is only a 12-hour timer, but it has an AM/PM bit so you can easily convert to 24-hour time as well. Much to the credit of the people at Commodore, the pin-outs of the 6526 are nearly identical to those of the 6522. Figure 1 shows the pin-outs for the 6526 and the 6522 where there is a difference. In a system which does not make heavy use of the hand-shaking functions, a 6526 is directly hardware-replaceable for a 6522. Also, most of the internal workings of the 6526 are nearly identical to the 6522. Many of the registers have the same names and functions, although different addresses. Table 1 is a listing of the register names and functions of the 6526.

I would like to concentrate on the time-of-day clock. This clock consists of four time registers with addresses \$08-\$0B for tenths of a second, seconds, minutes, and hours respectively. There are also two control registers, CRA and CRB (see table 2), which are used for initialization of the TOD clock and other functions. Each time register is written to and read out in BCD (binary coded decimal) which makes it easy for driving displays, but a slight problem for BASIC. The TOD clock requires an external TTL signal of 50 or 60 Hz to operate. The choice of 50 or 60 Hz is programmable by bit 7 of CRA.

Besides the TOD function there is also an alarm mode which allows an interrupt to be generated at any given time. The alarm time is written into the same registers that the TOD is written to, except that bit 7 of CRB is set to 1 for setting the alarm. CRB 7 set to 0 allows access of the time registers.

As previously mentioned, proper function of the TOD clock requires an external TTL level clock on pin 19 (labelled TOD). The TOD pin on the 6526 is where CB2 is on the 6522. There are a number of ways to generate a reference signal for the TOD clock: 1. divide the processor clock down with counters, 2. use the 16-bit counters on the 6526 to divide the system clock, and 3. pick off the 60 Hz AC line voltage and convert it to TTL levels. I don't like number one because it involves adding too many extra chips to my system. Number 2 wastes the counters in the 6526, which are more useful in other applications. Therefore, I have chosen number 3.

You'll see two ways to implement a 60 Hz clock for the AC line in figure 2. I use the circuit in figure 2a since I have a transformer in my system with a secondary voltage less than 60V peak-to-peak (the limit for the inputs of the 1489 receiver). This circuit works because all of the power supplies in my OSI system have a common ground. If you don't have a spare 1489 in your system, the circuit in figure 2b will work just as well. Here you are not as limited to input voltages; just pick the resistor value that keeps the current into the base of the transistor and diode within the limits for the components used. Almost any transistor will work. I use a 2N2222. Whatever method you use, make sure you *never* connect directly to the 100V AC lines. Use a transformer or opto-isolator. You will keep yourself and computer from an untimely end.

To provide a clearer idea of how to use the 6526, I have included a simple program written in BASIC (see listing 1). This program lets you set the time of day and display the time in an endless loop. It merely serves as a guide to set up the 6526 in the time mode. To achieve the proper setting and reading of the time registers, the Hours register must be written to or read first. On a write to Hours the TOD clock is stopped and not restarted until there is a write to the Tenths of a Second register. This assures that the clock starts at the intended instant. Reading from the Hours register causes all data to he latched until the Tenths register is read. If it is not necessary to read the hours, the other registers may be read but the data will not be latched.

The 6526 is so easy to use that I was able to unplug my 6522, plug in the 6526, and make the necessary connections in about ten minutes. Shortly thereafter I had a real system TOD clock complete with interrupts. Not only do I have a TOD clock in my system with no processor overhead to keep track of the time, but I have also retained all of the major functions of the 6522 which I was previously using. Not bad for the few hours I invested to hring it up.

At the time I wrote this article, the 6526 was not yet widely available. The price should be about $10 for the 1 MHz version.

Martin DeGeorge may be contacted at Threshold Technology, 1829 Underwood Blvd., Delran, New Jersey 08075.

Listing 1

```
10 REM 6526 ROUTINES
20 CIA=63232 :REM BASE ADDRESS OF 6526 = $F700
30 POKE CIA+2,00 :POKE CIA+3,00 :REM SET ALL I/O AS INPUT
40 REM
50 REM SET UP TIME OF DAY CLOCK
60 POKE CIA+14,0 :REM 60 HZ MODE
70 POKE CIA+15,0 :REM TOD ALARM OFF
80 INPUT "ENTER TIME OF DAY HH,MM,SS ";HH,MM,SS
90 IF HH>24 THEN PRINT "IMPROPER HOURS": GOTO 80
100 UH=0: PM=0
110 IF HH>11 THEN HH=HH-12 : PM=1 :REM CHECK IF AFTER NOON
120 IF HH>9 THEN HH=HH-10:UH=1:REM IF HOURS>10 SPLIT 10'S & 1'S
130 HH=128*PM+16*UH+HH: REM SET PM BIT IF AFTER NOON PM=BIT 7
140 IF MM>60 THEN PRINT "IMPROPER MINUTES": GOTO 80
150 REM BREAK MINUTES INTO 10'S AND 1'S
160 MM=MM/10
170 UM=INT(MM) :REM 10'S OF MINUTES (UPPER NIBBLE)
180 XN=(MM-UM)*10
190 XN=XN+.00000001
200 LM=INT(XN) :REM 1'S OF MINUTES (LOWER NIBBLE)
210 MM=UM*16+LM :REM MAKE UM AND LM NIBBLES INTO BYTE
220 REM
230 IF SS>60 THEN PRINT "IMPROPER SECONDS": GOTO80
240 SS=SS/10
250 US=INT(SS) :REM 10'S OF SECONDS (UPPER NIBBLE)
260 XN=(SS-US)*10
270 XN=XN+.00000001
280 LS=INT(XN) :REM 1'S OF SECONDS (LOWER NIBBLE)
290 SS=US*16+LS :REM MAKE US AND LS NIBBLES INTO BYTE
300 REM
310 REM PUT VALUES INTO 6526
320 POKE CIA+11,HH
330 POKE CIA+10,MM
340 POKE CIA+9, SS
350 POKE CIA+8, 00 :REM TENTHS WHICH START CLOCK
360 REM
370 REM READ OUT CLOCK
380 HH=PEEK(CIA+11) :REM READ HOURS - LATCH TIME REGISTERS
390 MM=PEEK(CIA+10) :REM READ MINUTES
400 SS=PEEK(CIA+9)  :REM READ SECONDS
410 TS=PEEK(CIA+8)  :REM READ TENTHS OF SECONDS
420 TH=0:TT=0
430 IF (HH AND 128)>0 THEN TH=1:REM CHECK PM BIT 1=AFTER 12:00
440 IF (HH AND 16)>0 THEN TT=1:REM HOURS > 10 ?
450 HH=12*TH+10*TT+(HH AND 15):REM ADD ALL HOURS
460 REM CONVERT MINUTES
470 UM=MM AND 112:REM MASK OUT 10'S OF MINUTES
480 UM=UM/16 :REM CONVERT 10'S OF MINUTES
490 LM=MM AND 15 :REM MASK OUT 1'S OF MINUTES
500 MM=UM*10+LM :REM ADD 10'S *10 + 1' OF MINUTES
510 REM CONVERT SECONDS SAME AS MINUTES
520 US=SS AND 112
530 US=US/16
540 LS=SS AND 15
550 SS=US*10+LS
560 TS=TS AND 15 :GET TENTHS OF SECONDS
570 PRINT HH;MM;SS;TS
580 GOTO 380 :REM ENDLESS LOOP
590 END
```

Editor's Note: The value assigned to CIA in line 20 applies to the author's system. Use a value appropriate for your installation.

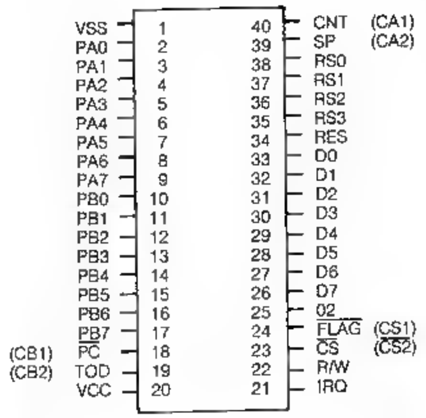


Figure 1: **Pin Configurations for 6526. (Where they differ, functions for the 6522 are shown in parentheses.)**

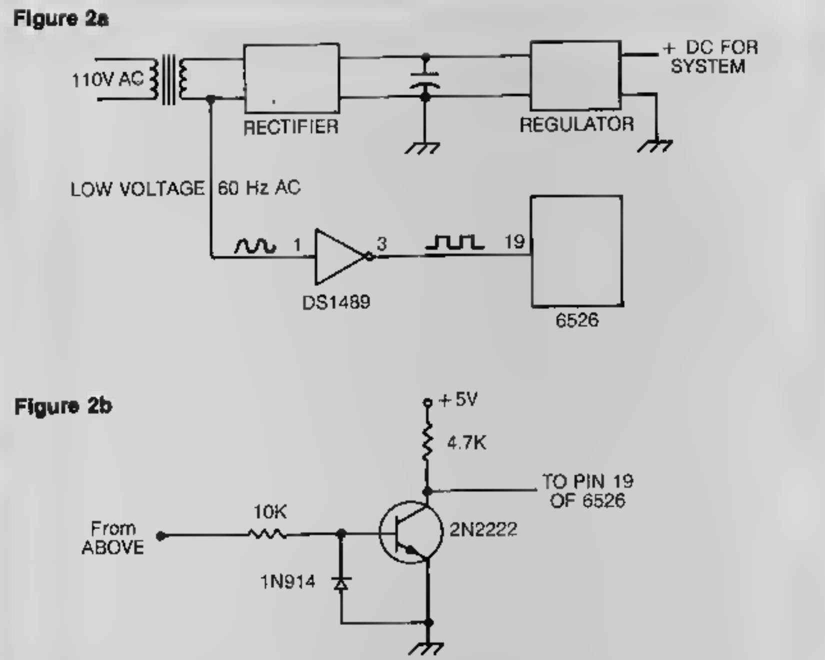


Figure 2: **60 Hz TTL Signal Generation from Low Voltage ac.**

Table 2: **Control Register Bit Actions**

Bit Position	Name	Function for bit = 1 or 0
CRA7	START CLOCK	1 = start timer A 0 = start timer B
CRA6	SPMODE	1 = serial output on pin 39 0 = serial input on pin 39
CRA5	INMODE	1 = timer A counts on negative transition of 39 0 = timer A counts on 02 pulses
CRA4	LOAD	1 = force load of timer prescaler 0 = no effect
CRA3	RUN MODE	1 = timer countdown to 0 generates interrupt and stops 0 = timer countdown to 0 generates interrupt and continues
CRA2	OUT MODE	1 = toggle of output to port B on timer underflow 0 = pulse output of port B on timer underflow
CRA1	PBON	1 = timer A output to PB6 0 = PB6 normal I/O
CRA0	START	1 = start timer A 0 = stop timer A
CRB0-CRB4		Similar function as CRA0-CRA4 for timer B except CRB1 which controls timer B out to PB7
CRB5,6	INMODE	Bits on CRB5 and CRB6 select input mode of timer B
CRB7	ALARM	1 = set alarm time on write to TOD registers 0 = set TOD on write to TOD registers

CRB6	CRB5	Timer B Counts On:
0	0	02 pulses
0	1	negative transitions on CNT
1	0	timer A underflow
1	1	timer A underflow while CNT = 0

MICRO

Programmable Reverse Video for the C1P

by Charles L. Stanford

Thls articla shows you how to add programmable reversa video to your C1P. The modification Involves some programmlng, and adding circuitry to your computer.

REVERSE VIDEO requires:

OSI C1P or Superboard (600 Board)

The reverse video option requires modification to your C1P, some additional circuitry and some software. This modification requires above-average skills in electronic construction, as well as substantial programming ability. While I've tried to make the actual changes on the main board as easy and risk-free as possihle, it's still very close to the equivalent of minor hrain surgery on your best friend.

OSI's Video System

Unlike many other machines, the C1P video refresh is completely hardware-based. In other words, the microprocessor devotes no time or effort toward keeping a proper display on the screen, but modifies the video RAM only when required to do so by the program. As a result, the video display has no undesirahle streaks caused by software timesharing. We are, however, unahle to make relatively simple program changes to achieve full control of the image.

Programmable Reverse Circuit Description

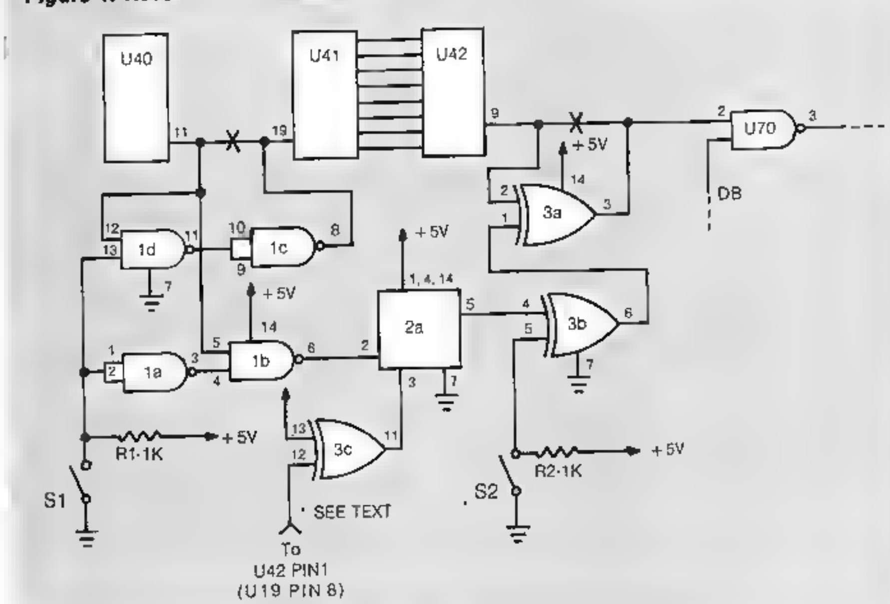
The circuit is relatively simple. It requires only three chips, can fit on a very small add-on board, and allows you to convert your computer back almost instantly to its original hardware configuration. It does cost a little in lost versatility: the upper 128 graphics characters are "lost" to use wbile the video reverse switch is closed. I have found that to be no inconvenience since we generally use the reverse video to enhance programs that use alphanumerics only.

The add-on circuit primarily consists of three elements: the detecter, the latch, and the inverter. The detecter is connected, in series, with the most significant hit of the video data. As shown in figure 1, NAND gates 1b and 1d each detect the status of the bit. Treatment of the bit is also conditioned by the status of switch S1. IC1d either inverts it or ignores it; IC1b either detects it or ignores it. If S1a is open, the bit is passed along through IC1c and appears unchanged to character generator U41. Likewise, IC1b ignores it and its output remains high. IC2a, half of a dual-D flip-flop, acts as a latch. It is clocked by the same latching signal used by U42, the parallel-serial shift register, and retains the status throughout the time needed to send one character to the screen.

The inverter uses two gates of a very versatile IC — the 7486 "exclusive OR" chip. In this circuit, it acts as both an inverter and a non-inverting gate. IC3a passes the serial video signal unchanged as long as pin I is held high, but pulling that pin low causes the signal to invert! In a similar manner, IC3b is used to condition the signal from the detecter and the latch circuits. Holding switch S2 high allows the signal from the latch to pass. Closing the switch inverts the output, effectively causing the image to he inverted constantly.

The net result of this circuit is to allow four conditions. When both

Figure 1: Reverse Video Modification



switches are open, the computer acts normally. Closing S1 inverts those characters which have a "1" in the left-most bit position (bit 7). Closing S2 inverts the entire screen. Closing both causes the characters which have bit 7 high to be normal, and the remainder to be inverted.

As I mentioned before, the price of this reverse video capability is the loss of the top 128 graphics characters. As long as switch S1 is open, the entire 256-character font of the character generator ROM is available. But closing that switch causes any character with a code greater than 127 (7F hex) to have the most significant bit detected and changed to low. Then the lower 128 show up on the screen normally, and the upper half show up as their inverted complements. For example, POKEing the graphics character 51 ($33) to a screen location will cause the character "3" to appear. POKEing the character 179 ($B3) with switch S1 closed will cause an inverted "3" to show. Essentially, the top bit is checked, stripped off, and changed to "0". If the same sequence is performed with S1 open, the graphics character normally corresponding to 179 will appear.

Modifying the 600 Board

Since I am always leery of damaging the PC board while making additions and modifications, I used an "add-on" board for this project. In addition, I devised a plug-in method that restores the main board almost instantly to its original configuration. As shown in figure 1, only two traces on the main board need to be cut. These are marked by an "X". Then wires are run from either side of the cuts to prototype socket U28. By connecting the leads as shown in figure 2a, a properly jumpered DIP header can be used as a shunt in place of the plug from the add-on board, restoring normal operation.

Start by installing a 16-pin solder-tail IC socket at U28. Be sure to use a low-wattage pencil-type iron, and practice on an old board if you're rusty. Next, cut the traces. It's best to use a jeweler's loupe or other magnifying lens, and carefully scratch away about 1/8 inch of the trace with a sharp knife blade. First, cut the line on the top of the board (component side) between U40 pin 11 and U41 pin 19. It starts at U40, but soon runs under U41's socket. Cut it about ¼ inch from pin 11 of U40.

Now, find the trace that leaves U70 pin 2 and heads for the keyboard. It only runs one inch before passing through the board. (Remember the location of this plated-through hole. It will be used later.) The trace now runs on the bottom toward the right, and again passes through to the top. It runs from there toward the front again, ending at U42 pin 9. Cut the trace on the bottom of the board near the hole by U70.

Figure 2a: **Socket U28 — Top View**

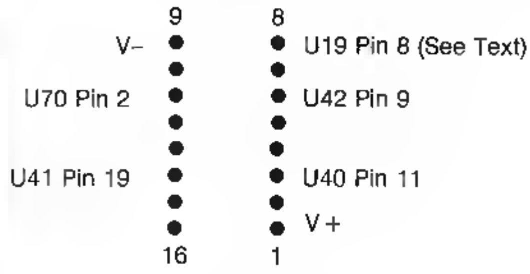
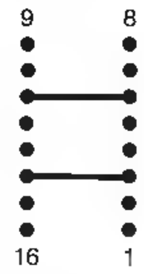


Figure 2b: **Jumper header for U28 — Top View**



Next, connect the socket at U28. Using fine-gauge insulated wire, connect each pin as shown in figure 2. It's easier to connect U40 and U41 by slipping the wire down into the sockets at the proper pin, than to try to solder to the small bit of PC board trace showing. If necessary, remove the ICs, exercising great care. For the other jumpers, use the two holes where the trace passes to the bottom of the board for your wire connections. Note that a connection to U42 pin 1 is marked "see text." I suggest that you use figures 1 and 2 as they appear until the new display reveals timing problems serious enough to require the fourth IC shown in figure 3. So for now, hook U42 pin 1 (which also connects to U19 pin 8) to U28 pin 8. Connect the positive and negative buses to pins 1 and 9, respectively.

Finally, solder jumper wires across a 16-pin DIP header as shown in figure 2b. Install the header, and try your computer. It should work normally. If not, troubleshooting should be easy since you've only made minimal changes.

Building the PC Board

Several techniques can be used to build your board. In this case, wirewrap is probably the best option. Equipment and supplies are readily available and are easy to use. It is essential to use a check list or schematic, and carefully check all connections when finished. Check the board under power first *without* ICs, and then *with* ICs, and measure current drain with a good volt/ohmmeter. Insert the ICs correctly. These TTL ICs will take a lot, but they cannot stand even a short period of inverse voltage, so make sure they get inserted properly.

The switch(es) can be mounted on your keyboard near either the left or right rear (just below the nameplate). When drilling, be very careful not to

Figure 3a: **Piggybacking ICs**

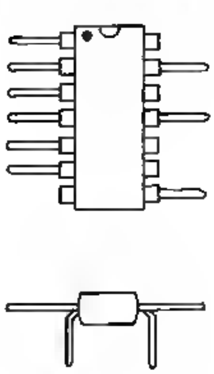
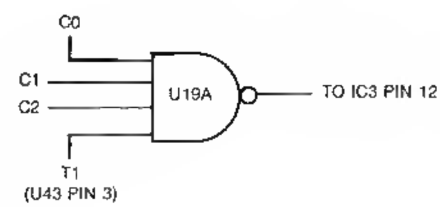


Figure 3b: **Connections for Optional IC**



mar the finish or get metallic cuttings in the works. Use stranded insulated wire to connect the small board with the switch and on the second IC header. It's not a bad idea to use some sort of socket/plug in the leads to the switch if you expect to disassemble your machine very often; it cuts down the stretching and bending of the wires.

Testing the Add-On

Have the TV or monitor warmed up before the computer is powered. Then, if the screen doesn't show a reasonable display, turn the power off immediately and check all wiring very carefully. Using an ohmmeter, make sure every point is properly connected to, and *only* to, the proper other points.

Listing 1

```
10  REM -VIDEO REVERSE DEMO
20  INPUT "ENTER A STRING";X$
30 A$ = X$: GOSUB 220:X$ = A$
40  PRINT X$
50  INPUT "ENTER A NUMBER";X
60 A = X: GOSUB 200:X$ = A$
70  PRINT X$
99  END
200  REM -REVERSE NUMBERS
210 A$ =  STR$ (A)
220  REM -REVERSE STRINGS
230 B$ = "": FOR X = 1 TO  LEN (A$)
240 C$ =  CHR$ ( ASC ( MID$ (A$,X,1)) OR 128)
250 B$ = B$ + C$: NEXT X
260 A$ = B$: RETURN
```

Listing 2

```
                  ;REVERSE VIDEO ROUTINE
                  ;
                  ;BY CHARLES STANFORD
                  ;
                  CTRLI    EPZ $09         ;CONTROL I CHARACTER
                  LF       EPZ $0A         ;LINE FEED
                  CR       EPZ $0D         ;CARRIAGE RETURN
                  ESC      EPZ $1B         ;ESCAPE CHARACTER
                  BRANCH   EPZ $F7         ;LBLC + 1
                  ;
                  OUTPUT   EQU $FF69       ;MONITOR OUTPUT ROUTINE
                  GETCHR   EQU $FFBA       ;GET CHARACTER ROUTINE
                  ;
                           ORG $D8
                  ;
00D8 20 BA FF              JSR GETCHR      ;GET A CHARACTER
00DB C9 09                 CMP #CTRLI      ;IS IT A CONTROL-I?
00DD D0 05                 BNE LBLA
00DF A2 00                 LDX #$00        ;IF YES, MODIFY BRANCH
00E1 86 F7                 STX BRANCH      ; TO REVERSE CHARACTERS
00E3 60                    RTS
00E4 C9 1B        LBLA     CMP #ESC        ;IS IT ESCAPE?
00E6 D0 04                 BNE LBLB
00E8 A2 02                 LDX #$02        ;IF YES, RESET BRANCH TO
00EA 86 F7                 STX BRANCH      ; TO DISPLAY NORMAL CHARACTERS
00EC 60           LBLB     RTS
00ED C9 0D                 CMP #CR         ;IS OUTPUT CHAR A CR?
00EF F0 09                 BEQ LBLD
00F1 C9 0A                 CMP #LF         ;LINE FEED?
00F3 F0 05                 BEQ LBLD
00F5 18                    CLC
00F6 90 02        LBLC     BCC LBLD        ;BRANCH ALWAYS (MODIFIED ABOVE)
)
00F8 09 80                 ORA #$80        ;SET HIGH BIT ONLY IF CTRL-I
00FA 4C 69 FF     LBLD     JMP OUTPUT      ;TO MONITOR OUTPUT ROUTINE
00FD                       END
```

Listing 3

```
3000  REM -MACH LANG REVERSE VIDEO ROUTINE
3010  POKE 536,216: POKE 537,0
3020  POKE 538,237: POKE 539,0
3030  FOR M = 216 TO 252: READ D: POKE M,D: NEXT
3040  DATA 32,186,255,201,9,208,5,162,0,134,247,96,201,27,208,4
3050  DATA 162,2,134,247,96,201,13,240,9,201,10,240,5,24,144,2
3060  DATA 9,128,76,105,255
```

Since your machine will have been without power for some time, the RAM will probably be well-scrambled, and at least a few graphics characters will appear. Don't hit Break at this time; try the switches, and get a feel for the way they work.

This is also the time to look for any timing problems. Compare the reversed characters with the OSI *Graphics Reference Manual*. If the timing from U19 pin 8 is delayed too much by passing through ICs 2 and 3, the screen will reverse a bit late, and change back a bit late. Reversal of characters in a row will only be noticeable at the beginning of the first row and the end of the last. This phenomenon occurs when the signal from U42 is reversed just slightly out of sync with the latch trigger from NAND gate U19. Two solutions are possible. Since the cause of the delay is the extra gate transmission time in IC2a, IC3b, and IC3a, using faster gates will help. The very fast throughput of 74S-ICs reduces differential delay to the point that it is virtually unnoticeable on the screen.

The disadvantages here are extra cost, the difficulty of finding Schottky chips, and additional power drain. Since I couldn't wait for a mail-order delivery taking several weeks, another solution seemed practical — equalize the delay. This was done by installing another 74LS20 on top of U19 with all but pins 7, 9, 10, 12, and 14 bent out so they don't make contact. This is called "piggybacking" and is a neat and effective way to add additional circuits to an existing board.

As shown on the 600 board schematic, U19 uses the gating of C0, C1, C2, and T3 to trigger the latch in the parallel-serial shift register U42. T3 is merely the clock signal delayed through three gates to match delays already present in the video circuits. It's obvious that a lesser delay in the trigger to latch IC3 might even things out. Accordingly, U19A piggybacked to U19 can use three of the signals, and pin 13 can be connected to U43 pin 1, the T1 signal (clock with only one gate of delay). Use pin 8 of U19A instead of pin 8 of U19 to trigger latch IC2a. U43 has some solder pads that make connection of the jumper very convenient. To prevent damage to the ICs, be sure to put a dab of solder on each of the pins common to U19 and U19A. Again, a good magnifying glass is invaluable. Pins 1 through 6 are left unconnected.

When you test the computer again, carefully check the reversed characters

to be sure that they are completely in sync with the reversing circuit. You may find it necessary to use the clock itself, or T2, but T1 seems to be just about right.

Programming Techniques

There are at least half a dozen ways to use BASIC or machine language software to capitalize on your new character reversing capability. Using the CHR$, ASC, LEN, and MID$ functions, entire strings can be readily inverted by a relatively short and straightforward subroutine. The demonstration program in listing 1 can also be used in a game or financial planning program to highlight certain inputs or headings. Either inputs or internal strings will reverse, and numeric variables can also be reversed by using the STR$ function.

The machine language program in listing 2 is quite a bit more sophisticated. It can reside in the unused (by BASIC) RAM at the top of page zero, but remember that the monitor does use the space when you break. The program intercepts both the "character-get" and the "screen-write" routines of BASIC by changing the indirect addresses at $0218 and $021A. Then the data can be processed as needed for reverse video.

When the routine is in place, the first five lines get the character from the keyboard as usual, and only act if either the control-1 or escape key is detected. The control-1 causes the routine starting at $00E4 to force a "1" into the left bit of the character. Once the control-I is pressed, every character coming from either the keyboard or the ACIA will be inverted before being passed to the screen output or program storage. Hitting the escape key will return action to normal.

Notice that the routine is set to ignore carriage returns and line feeds. All other characters get the "reverse" treatment. Thus, be careful to use it only for those items which go to the screen or are within quotes. Trying to invert characters involved in program entry will badly confuse the BASIC interpreter, and lead to a program crash.

If you are familiar with the method Microsoft uses to store BASIC Source Code starting at $0300, you will be able to devise methods of actually changing the characters by modifying the program itself. Without going into details, it isn't too hard to write a BASIC program that will scan the source code for a particular line number, and then invert any characters between quotation marks within that line. I'm sure that you will find many creative ways to use this new capability.

Parts List

R1, R2 — 1KOhm ¼ watt
IC1 — 74LS00
IC2 — 74LS74 (option 74S74, see text)
IC3 — 74LS86 (option 74S86, see text)
IC4 — (optional — 74LS20)
S1, S2 — SPST miniature toggle switches (Radio Shack 275-324)
S1A — optional in place of S1 and S2 SPDT center off min toggle switch (Radio Shack 275-325)
Misc. — PC board, IC sockets, IC header, Molex connector, wire, etc.

MICRO

SDS GIVES YOU ALL YOU NEED!

APPLE-DOC

We are proud to announce an updated version of our best seller, **Apple-Doc** in DOS 3.3. This versatile package consists of three powerful reference utilities; **Variable Cross-Reference** which creates a table of every variable and the line number on which it occurs, **Line Number Cross-Reference** creates a table of every program lined called by a GOTO, GOSUB, etc., and all lines called from, and finally, **Constant Cross-Reference** creates a table of all numbers used in the program not listed by the above tables. Also included are **Replace**, a global replacement editor and **Lister** which produces a completely professional program listing. Additionally, the new 30 page tutorial manual makes **Apple-Doc** quick and easy to use. Whether novice or professional **Apple-Doc** more than pays for itself in time savings alone!

Price: $49.95 on disk

a.c.e.

Applesoft* Command Editor, or **A.C.E.**, combines a large number of the most useful programming utilities into one co-resident package. Co-resident means these routines become a working part of the Apple II*, thereby making **A.C.E.'s** various commands and utilities immediately available WHILE you're entering or editing existing program lines. It's functions include cross-referencing individual variable names, displaying current values of all variables, and information on memory and diskette status. For machine language programming there is a hex-decimal converter and all monitor commands can be executed directly from Applesoft. User defined macros allow strings to be input with a single keystroke and a powerful line editor makes editing easy and greatly reduces program development time. **A.C.E.** is a winner!

Price: $39.95 on disk

ListMaster

List Master, our latest addition, provides a number of utilities for performing large scale changes to your program listings in a high-speed and automatic way. **List Master** includes **Applespeed** which can remove REMarks, shorten variable names, combine lines, and renumbers by 1's in any Applesoft program. **Smart Renumber** not only renumbers but also gives you the option of preserving logical blocks of line numbers you have established for various routines thereby retaining the original logic and workability in the renumbered program. Also included are merge routines that allow you to quote blocks of lines from one part of the program to another and automatically renumber appropriate line references. **Comp-List** compares any two Applesoft or Integer programs and lists any lines added, deleted or changed.

Price: $39.95 on disk

All programs require 48K and Applesoft in ROM or language card.
Calif. residents add 6% sales tax.

*Apple II and Applesoft are registered trademarks of Apple Computer Co.

SDS

southwestern data systems

P.O. Box 582 • Santee, California 92071 • (714) 562-3670

Structured Programming in BASIC09

by Brian Capouch

The BASIC09 programming language, designed after the beginning of the microcomputer revolution, fully exploits the advantages to be gained from both structured programming techniques and user interaction. BASIC09 is available for the 6800/6809-based OS-9 operating systems.

Here where I live, in the heart of the country, my digital computer, with its busy little clock ticking away two million times in a second, provides an interesting counterpoint to the natural processes that go on around me. Many of these processes exist in geologic time that is measured in eons. In an attempt to teach the youth of my community to program computers, I encourage them to avoid getting their psyches all wrapped up in those 500-nanosecond ticks. Many programmers I know treat each upcoming software deadline as if it were the end of the world as they know it. I propose that programming should be approached in tbe casual manner of the old-time mechanic, who feels that sitting down under a tree and thinking about a problem for a few minutes every once in a while speeds the hand wben the actual work must get done.

It is certain that many points of common ground exist between practitioners of our local homey arts and computer programmers. All concerned find themselves engaged in hehavior that revolves around cyclic processes. Crops are planted and harvested, motors run, break down and are repaired, and, alas, the same programs are written, with minor variations, again and again. Farmers and mechanics have always seemed to me to be more inured to this cycling than programmers — and with good reason. The programmer should not be compared to the farmer but to the blacksmith, since his job is to create tools for others to use. It would be a poor blacksmith who blindly returned to his drawing-boards to design each successive plowshare from scratch.

Elements of Structure in Programs

To alleviate this problem, recent history has seen the rise of "structured programming." Although this term has been overused, it nonetheless suggests an attempt to program scientifically, and thus is to be welcomed.

Although there are almost as many definitions of the meaning of structured programming as there are practitioners of it, several points of agreement exist regarding its essential elements. The first is *modularity*. Decomposing a problem into its elemental parts makes it is easier to define separate problem-solving stages, or modules, for each problem segment. It is surprising that programmers have been so slow to acknowledge explicitly the value of this common sense approacb.

Modularity also has another virture, given a powerful enough language. Routine modules can he maintained in libraries and used as building blocks in programs. It is impossible to discuss this topic without mentioning a classic work in this field, *Software Tools*, by Brian Kernighan and P.J. Plauger. This hook takes tbe topic of modularity from thoery into practice, providing hundreds of examples of simple tools that can be combined into very complex and powerful programs. Their original book, which features examples written in the psuedo-language Ratfor, has been augmented by a recent edition in Pascal. I like to challenge my students who are just experiencing the heady transition from neophyte into programmer to "come hack and talk to me wben you can discuss *Software Tools.*"

Another important structuring technique is that of *blocking*, which simply means writing an easy-to-read program. More than one sage programmer has ranked readability as the most indispensable characteristic of top-quality code. It is tbis aspect of structuring that is most likely to be ignored by the beginner, as he reassures himself that "I'll be the only one reading this progam anyway, and I will be able to remember what it was I was doing." Beginners are almost always romantics, and after shooting himself in the foot often enough anyone will begin looking at either his gun or his trigger finger.

Both of these aspects of structured programming imply that computer programs must be regarded as tools and thus treated as capital goods that will be maintained, instead of expendable consumption items that will be used once and then discarded. I could handily retire on tbe wages paid each year to programmers to refamiliarize themselves with programs that they had thougbt finished at some earlier point in time. The hasis of structured programming, then, is the belief that software tools should be built in logical increments and that these modules should be self-documenting.

Man and Machine Interacting

An editorial column that appeared in the newspaper *Infoworld* last summer has provided me witb fodder for hours of introspection. That column spent several inches wondering about how the hardware practice of having terminal displays with 80-character lines had ever evolved. A number of tbeories were presented and examined before the writer admitted that he just plain didn't know. As I was reading, I noticed an IBM punch card grinning at me furiously from a box of old college mementos. "These kids," it seemed to say.

This anecdote points up a fact about the modern computer world that is every bit as novel as the microcomputer: The idea of real-time interaction between user and machine. My students find it as hard to believe my recollections of the "submit cards-get printout-change cards-submit cards" cycle that programming used to require, as they do my bragging about having to use a crank-style telephone during my youth. And all credulity vanishes when they find out that people still program like that today.

Microcomputers are set apart from previous computers by the accessibility they provide to an interactive experience. These topics are discussed thoroughly in a recent book by P.J. Brown entitled *Writing Interactive Compilers and Interpreters*. In it, Dr. Brown examines the concept of interactivity in general, and discusses how compilers and interpreters can be constructed to enhance interactivity. This book is interesting even if you are not planning to write a compiler. If offers incredibly keen insight into the process of programming.

Dr. Brown's observations are valuable in two respects. First, his advice on writing interactive processors is as applicable to applications programmers as it is to systems programmers. Second, he specifies what he considers to be an ideal, yet practical, interactive language.

Enter BASIC09

Almost every ideal function that Dr. Brown discusses in his excellent book is actually available in a real programming language, BASIC09. This language was written for the 6809 microprocessor by the Microware Corporation to run on their OS-9 operating system. Although I had been using BASIC09 for a year before I came upon Dr. Brown's book, and was thus well acquainted with the amazing power that the language possesses, it wasn't until I read his description of "ideals" that the rationale behind many of its features became fully apparent.

BASIC09 is more than a programming language, and it is certainly much more tban BASIC. It consists of an integrated package of processors tbat includes a multi-pass compiler, a powerful text editor, and a run-time debugger that is entered automatically on the generation of an error. Technically speaking, it is both a compiler and interpreter, as it compiles source code into an optimized I-code which is then interpreted by a run-time processor. Syntactic features give it the programming power found in many modern, highly structured languages, while it retains compatibility with almost all standard-syntax BASICs. What follows is a brief description of those features, although it is impossible within the space of this article to provide more than a whirlwind tour of its spectacular power.

Note that the operating system under which BASIC09 runs is every bit as advanced and wondrous as the language I am about to describe. Readers are referred to an earlier article in MICRO (42:81) for an overview of this multi-user, multi-tasking, Unix-like operating system.

Listing 1

```
PROCEDURE multiply
 0000       (* Program to demonstrate "EXITIF-ENDEXIT" and "LOOP-ENDLOOP"
 003D       (* Multiplies two real numbers input by user; prints product
 0079       DIM multiplier,multiplicand,product:REAL
 0088       PRINT "Enter numbers to be multiplied"
 00AA       PRINT "(Second number '0' to quit"
 00C8       LOOP
 00CA         INPUT multiplicand,multiplier
 00D3       EXITIF multiplier=0 THEN
 00E0         PRINT "It was nice working for you"
 00FF         PRINT "Goodbye"
 010A       ENDEXIT
 010E         product=multiplicand*multiplier
 011A         PRINT product
 011F       ENDLOOP
 0123       END
```

Listing 2

```
PROCEDURE powers
 0000       (* Procedure to demonstrate nested"IF-THEN-(ELSE)-ENDIF"construct
 0043       (* Takes input value to given power
 0066       DIM value,result:REAL
 0071       DIM power:INTEGER
 0078       PRINT "Program to print powers of real numbers"
 00A3       PRINT "Maximum=3; Enter '0' for power to quit"
 00CD       LOOP
 00CF         INPUT "Enter value ",value
 00E3         INPUT "Enter power ",power
 00F7       EXITIF power=0 THEN
 0103         PRINT "Nice working for you--goodbyte!!"
 0127       ENDEXIT
 012B         IF power=1 THEN
 0137           result=valuc
 013F         ELSE
 0143           IF power=2 THEN
 014F             result=value*value
 015B           ELSE
 015F             IF power=3 THEN
 016B               result=value*value*value
 017B             ELSE
 017F               PRINT "ILLEGAL VALUE!/!"
 0193               result=0
 019B             ENDIF
 019D           ENDIF
 019F         ENDIF
 01A1         PRINT result
 01A6       ENDLOOP
 01AA       END
```

Listing 3

```
PROCEDURE getname
 0000       (* Demonstrate complex data types
 0021       (* Input data into a complex name-address structure
 0054       TYPE item=name,address(2):STRING[40]; zip:REAL
 0073       DIM record:item
 007C       PRINT "Please enter items as requested"
 009F       PRINT "Enter 'RETURN' for name to end session"
 00C9       LOOP
 00CB         INPUT "Enter name ",record.name
 00E2       EXITIF record.name="" THEN
 00F1       ENDEXIT
 00F5         INPUT "Line 1 Address ",record.address(1)
 0112         INPUT "Line 2 Address ",record.address(2)
 012F         INPUT "Zip Code ",record.zip
 0144         RUN displayname(record)
 014E       ENDLOOP
 0152       END
```

Listing 4

```
PROCEDURE convert
 0000       (* Example of implicit type conversion
 0026       (* Converts input string into equivalent ASCII decimal codes
 0062       TYPE simple=item:STRING[32]
 0072       TYPE complex=ascii_code(32):BYTE
 0082       DIM first:simple
 008B       DIM second:complex
 0094       PRINT "This procedure converts strings to decimal ASCII values"
 00CF       INPUT "Enter a string <32 characters ",first.item
 00F9       second=first
 0101       FOR index=1 TO LEN(first.item)
 0118         PRINT second.ascii_code(index); " -";
 012A       NEXT index
 0135       PRINT
 0137       END
```

Syntactic Features

Syntactically, BASIC09 is a hybrid language. Based on BASIC, it borrows many structuring elements from Pascal. For instance, the following is a legal BASIC09 program:

```
0010 PRINT "ENTER NUMBER OF
     TIMES TO LOOP"
0020 INPUT A
0030 FOR I=1 TO A
0040 PRINT "CRETIN LOOP PASS NO. ";I
0050 NEXT I
0060 END
```

In this simple example, which all BASIC programmers should understand, the user inputs a number which is then used to control the execution of a loop. Two variables are used, both of which, since they are not explicitly defined, are of the real or floating-point data type. This conforms to standard BASIC programming practice. String variables, with a default length of 32 characters, are defined similarly by appending a dollar sign ("$") to a variable name. However, other types of data are allowed in BASIC09, those of *byte*, *integer*, and *Boolean*. Variables of these types must be explicitly allocated using the "DIM" statement. In the example program listed above, if we assume that the user will keep his request to a quantity that can be stored as a signed integer (+32767 to −32768), we can take advantage of integer math routines and make execution of our program much faster. Another significant gain can be realized by ommitting line numbers. They are not required by BASIC09, and are wasteful of program memory space. We can re-do our program, explicitly dimensioning our data types, and jettisoning the line numbers:

```
DIM loopindex,topcount:INTEGER
PRINT "Enter desired number of passes "
INPUT topcount
FOR loopindex=1 TO topcount
      PRINT "Smarter loop pass No. ";
      loopindex
NEXT loopindex
END
```

In this version of the program, further features of the language also appear. One nice protocol that we have adopted is to use descriptive names for our variables, and to always keep them in lower case. This is because the BASIC09 "decompiler" automatically capitalizes keywords when a source program is listed. If you keep variable names in lower case, they become easy to distinguish. This helps fulfill our structuring goal of making programs self-documenting. The listing above also displays BASIC09's automatic "prettyprinting." This facility, which indents program lines according to their logical hierarchy, provides an easy way to grasp program structure, and aids debugging.

From this point forward all of our examples will be actual output by BASIC09's listing mechanism. Two features bear some explanation. First, the hexadecimal numbers on the left-hand side represent the relative *I-code addresses* into which the corresponding program source lines compile. They show the programmer the amount of memory being consumed by his program, and serve as pointers into the compiled code for tracking down errors during the debugging process.

Procedure Orientation

Another feature of the language seen in our examples is its *procedure* organization. BASIC09 allows programs (called procedures) to call other procedures by name, and allows them to be separately compiled — a feature lacking even in standard Pascal. This permits users to build libraries of procedures that perform standard and often-used functions, which is an important step toward the modularity requirement for structured programs. Parameters can be passed to procedures in much the same manner as in Pascal, which is to say both by reference (by using the name of a variable), and by value (by using a constant value or expression). Thus, in the manner advocated by Kernighan and Plauger, procedures can "hide" the details of their operation from other procedures that call them. Therefore, data linkage is loosely done through easy-to-spot, explicit parameters.

Loops and Conditional Statements

Loops in BASIC09 can be done using the familiar FOR-NEXT duo, the Pascal loops of WHILE...DO and REPEAT...UNTIL, or a loop-forever construct called LOOP...ENDLOOP. Any of these loops may be exited in a gentlemanly fashion by using the conditional EXITIF statement. The example procedure "multiply" uses the loop-forever construct, printing a "goodbye message" when the user has finished using the program's logic. (See listing 1. Note that the first two lines in the program listing are remarks, which can be signified using the "(*" characters as the first characters in a line.)

The full complement of looping structures allows the BASIC09 programmer to use the loop that will get the job done, and, at the same time, adds structure to his code.

Other logical features adding to BASIC09's power are two conditional branching statements: a "meat and potatoes" IF-THEN- (line number), and a structured IF-THEN-(ELSE)-ENDIF construct. The latter is indented in listings for logical clarity and will enable most programs to be written entirely without line numbers. IF statements can be nested to any required depth so that complex state selections can be made. The procedure "powers" demonstrates a four-way branch on an input value. (See listing 2.)

Data Type Definition

Again borrowing from Pascal, BASIC09 allows programmers to define unique data types built up from the "atomic" standard data types mentioned above. These user-defined types may themselves be part of further type definitions, and so on, forever. Thus arbitrarily complex, non-rectangular types may be constructed to fit the nature of data at hand. Advantages of this method include mnemonic naming of fields in a complex type, elimination of array-index calculation at run time, and simplified passing of parameters to outboard procedures and I/O routines. The procedure "getname" (see listing 3) illustrates the principles of complex typing. It calls a mythical procedure called "displayname" (not shown here) that prints name and address information on a line printer.

Implicit Type Conversion

Complex data types possess another significant attribute, although it could be argued that it belongs in the "giving razors to the baby" class. Data stored in complex type variables may be transferred to other complex variables of equal size with a simple assignment operation, regardless of the makeup of the respective types. This means type conversions can be done as simply as typing "=". For example, the procedure *convert* converts a string into its equivalent ASCII code values and displays those values. (See listing 4.) This listing is supplemented by a sample run.

Implicit type conversion is a built-in method of accomplishing things that were formerly done only with much anguish on the part of programmers. As with all extremely powerful tools, it is a double-edged sword, and must be used with caution.

When math is performed using variables dimensioned to different numeric

types, type conversion is done automatically by BASIC09's math routines to correspond with the data type of the variable receiving the result. Note that overflows and underflows for integer and byte data types do not result in an error — they merely wrap around zero. Programmers should therefore be careful to make sure they anticipate numeric ranges carefully, lest an unanticipated variable value lead to mysterious results at some unforeseen point in the future.

BASIC09's syntactic repertoire also includes full Boolean logical operations, bit twiddling, standard transcendental functions, and an extremely powerful PRINT USING function.

The Procedure Editor

Program development is by nature a cyclic process. In most compiled languages an outboard text editor is used to assemble program statements, which are then run through the compiler. Then, if bugs are found, the text editor must be reloaded, changes made, and another compile cycle initiated. This process continues until the program is debugged and running properly. This begins to sound like the old "punch-submit-debug-punch" routine, because of the two-stage nature of interacting with the computer.

To circumvent the problem, BASIC09 employs two interrelated techniques. The first of these is *incremental compilation*, which means that each line is compiled from source at the time that it is entered. In this way, most syntactic errors can be immediately detected and reported to the user. This precludes the annoyance of a simple typographical mistake slipping undetected through to the compiler. Lines containing these errors are also marked so that procedures containing them can not be run. This avoids the delay caused by the run-time system processing "good" lines as it ambles towards a syntactic error lurking deep within a procedure.

The second tactic employed by BASIC09 to decrease program development delay is to incorporate a text editor into the compiler. The two-stage process now becomes integrated, interactive, and much faster. The editor is both content and line-number oriented, so that the program segments can be accessed whether or not they fall within a numbered line.

Editor commands, which can apply either locally (i.e., to the currently displayed line) or globally (to all lines) include: search, change, list, delete, and renumber, and there are also commands to position the edit pointer within a procedure. The convenience of using this procedure editor has kept me consistently using BASIC09 in favor of Microware's powerful ISO-standard Pascal, because of the tremendous time savings it brings to program development.

User Workspace

BASIC09 employs what is called the "workspace" concept for managing user memory. At the time a user logs onto the system, he is assigned a workspace of arbitrary size. All procedures that exist in the source code form are required to reside in a user's workspace, where they are maintained by BASIC09 and its associated processors. Information is available to the user at all times regarding the quantity of program memory in use, the amount of data memory required by his programs, and the amount of remaining workspace. A typical workspace directory is given in figure 4 for the procedures listed above. It lists the procedures currently resident within the workspace, along with their memory requirements (in decimal). These requirements pertain to the source code; I-code is more compact. The asterisk ("*") marks the "current working procedure," which is accepted as a default argument by commands such as those that control disk I/O.

Debugging

An integral debugger, entered whenever the run-time processor detects an error, provides the final link in the BASIC09 program development chain. While in this mode, values of all variables can be displayed or changed, and the currently running procedure can be listed, as can the "procedure stack" or list of currently invoked procedures. While in this mode a tracing command can be employed to begin displaying each line as it is executed. A single-step command can execute statements one at a time.

Program flow can be interrupted by the programmer at any point in his source code by the addition of the "PAUSE" statement. This statement causes processing to stop and the debug mode to be entered. At this time any of the operations mentioned above can be performed, and the program resumed by typing "CONT". This function gets my nomination as the most valuable single feature of BASIC09; it is a painless way to debug complex code in easy stages.

Packing Procedures

Once a procedure has been written and debugged, there is no logical reason for the system to allocate memory for full variable names, comments, and other space-hogging constructs that are not germane to its actual running. Towards this end an optional extra pass of the compiler may be generated, packing the procedure to remove them. Once this has been done a procedure can be loaded into system memory *outside of the user workspace,* therefore making it available for multiple users *via* the OS-9 timesharing system. The only workspace memory overhead for this procedure then becomes the data memory required, which obviously cannot be shared safely by all users.

As an additional bonus, procedures which have been packed cannot be edited or listed, which means that for all practical purposes their source code is inaccessible. This can be very important to software developers who cannot afford wanton copying of source code. However, let me advise potential users to *always be sure you have a source code copy of a procedure already saved on disk* before *invoking the packing pass!* Otherwise even the programmer is locked out from his own source code.

Conclusion

I have illustrated those features of BASI09 which I believe make it excellent for the construction of applications tools. It provides the means for a programmer to systematize his undertakings so that he is not constantly writing the same code again and again. Structure provided by the language replaces structure provided by the programmer, freeing him for the more rewarding tasks of problem analysis and daydreaming. Readers who are interested in learning more about BASIC09 should contact Microware and order a programmer's manual, which contains a complete description of the language as well as numerous source code examples.

References

1. Amsbury, Wayne, *Structured BASIC and Beyond.* Potomac, Maryland: Computer Science Press, 1980.
2. *BASIC09 Programming Manual.* Des Moines, Iowa: Microware Corporation, 1981.
3. Brown, P.J., *Writing Interactive Compilers and Interpreters.* New York: John Wiley and Sons, 1979.
4. Capouch, Brian, "OS-9 and the 6809: Revolutionary Tools." *MICRO 6502/6809 Journal,* November, 1981, p.81.
5. Kernighan, Brian and P.J. Plauger, *Software Tools.* Reading, Mass.: Addison-Wesley, 1976.
6. Kernighan, Brian and P.J. Plauger, *Software Tools in Pascal.* Reading, Mass.: Addison-Wesley, 1981.
7. *OS-9 System Programmer's Manual.* Des Moines, Iowa: Microware Corporation, 1981.
8. *OS-9 User's Guide.* Des Moines, Iowa: Microware Corporation, 1981.

Write to the author at R.R. 2, Box 525, Wheatfield, IN 46392.

MICRO

Extensions to the C-Bug Monitor

by Ralph Tenny

Two veluable debugging functlons ere added to CBUG, e mechine lenguege monitor for the Color Computer. In addltion, the feetures of CBUG are summarlzed.

These Utilitles require:

TRS-80 Color Computer
CBUG monitor, on cassette or in ROM

Although the TRS-80C is an excellent, low-cost computer, Radio Shack originally offered no hope for expansion beyond the machine's ohvious uscs. However, Percom Data Company, Inc. has already produced an adapter which expands the TRS-80C by using standard SS-50 bus components. Other manufacturers will also undoubtedly support the machine.

As users dig further into the Color Computer's guts, they'll find that many internal provisions have heen made for expansion, both in the hardware and software.

The Micro Works, Inc., of Del Mar, California, brought out CBUG, an assembly language utility for the TRS-80C. The two programs included here are both supported hy CBUG. The first program, BKP, is entered using CBUG's J command, which transfers control to BKP long enough to type in a location where you want the breakpoint to happen. (If you haven't used a breakpoint program before, note that this address must point only to an op-code, and not to an operand or a data byte.) BKP then grabs the code pointed to, saves it, and replaces it with the 6809 opcode for SWI2. This code forces a full interrupt, stacking the entire machine contents and fetching an interrupt vector from $FFF4. Sincc the

Listing 1

```
                   *CBUGP:COPYRIGHT 1981 BY RALPH TENNY
                   *REVISED 6/81 BY RALPH TENNY
                   *REFERENCE: CBUG, COPYRIGHT 1981 BY
                   *THE MICRO WORKS, INC.
                   *THIS PROGRAM SEGMENT SUPPLEMENTS CBUG, GIVING IT
                   *BREAKPOINT, PAICH, CHARACTER SEARCH, AND KEYBOARD
                   *DECODING CAPABILITY, PLUS OTHER HANDY SUBROUTINES.

                   *****REFERENCES FROM CBUG*****

                   *NOTE: CHECK THESE LOCATIONS WITH THE VERSION OF
                   *CBUG THIS EXTENSION WILL BE USED WITH!

              0088 CURPTR  EQU    $88       CURSOR POINTER
              00FB PARAM   EQU    $FB       BUFFER FOR OUTHEX
              05FF SCREND  EQU    $05FF     END OF OISPLAY BUFFER
              0618 ENIRY   EQU    $618      IN TO CBUG INNEY
              0627 HEX     EQU    $627      IN TO CBUG OUTHEX
              0651 BADDR   EQU    $651      READ BINARY ADDRESS
              0661 BYTE    EQU    $661      GET A BYTE FROM KEYBOARD
              067F PCRLF   EQU    $67F      PRINT CRLF
              0680 PDATA   EQU    $68D      PRINT A STRING
              06AE OUTHEX  EQU    $6AE      PRINI A BYTE
              06B0 OUTS    EQU    $6BD      PRINT A SPACE
              077D HSTART  EQU    $77D      HARD START ENTRY
              07A5 INT     EQU    $7A5      INTERRUPT ENTRY
              07A9 WARMS   EQU    $7A9      WARM START ENIRY
              07C9 GETAOR  EQU    $7C9      GET TWO ADORESSES
              07D0 ECHO    EQU    $700      GET A KEY AND ECHO TO SCREEN
              07D9 REG     EQU    $7D9      PRINT STACK

                   *****EQUATES AND BUFFER DEFINITIONS FOR CBUGP*****

0010                       ORG    $0010
0010 00            DPLCH   FCB    0         STORAGE FOR DISPLAY CHARACTER
0011 00            MEMLIN  FCB    0         COUNTER FOR MEMORY CHARACTERS
0012 00            TXICHR  FCB    0         BUFFER FOR INPUT CHARACTERS
0013 00            SPARE   FCB    0         RESERVE BUFFER
0014 0000          TXTEND  FDB    0         END OF TEXT BUFFER
0016 0000          MEMPTR  FDB    0         NEXT MEMORY LOCATION
0018 0000          DXBFR   FDB    0         TWO-BYTE SPARES
001A 0000          DDBFR   FOB    0
001C 0000          DYBFR   FDB    0
001E 0000          OUBFR   FDB    0
0020 0000          DSDFR   FDB    0
              0103 SWI2    EQU    $103      VECTOR FOR SW12
              0100 SWI3    EQU    $100      VECTOR FOR SWI3
0D30                       ORG    $030      NEXT TO CBUG END

                   *ENTER HERE FROM CBUG USING 'J' COMMANO

0D30 34   40       GOIN    PSHS   U         SAVE CBUG ENVIRONMENT
0D32 36   36               PSHU   A,B,X,Y   HERE, TOO
0D34 17   F948     IN2     LBSR   PCRLF     RESET THE DISPLAY
0D37 30   8D 0193          LEAX   XPMPT,PCR FIND THE PROMPT
0D3B 17   F94F             LBSR   POAIA     AND PRINT IT
0D3E 17   FA8F             LBSR   ECHO      AND GET A KEY
0D41 1F   89               TFR    A,B       SAVE THE DATA
0D43 17   F977             LBSR   OUTS      PRINT A SPACE
0D46 30   8D 0011          LEAX   XTBL,PCR  FINO LOOKUP TABLE
0D4A 86   08               LOA    #IABND/3  COMPUTE TABLE LENGTH
0D4C E1   84       SEEK    CMPB   ,X        FIND THE COMMAND
0D4E 27   05               BEQ    GOIIT     IHERE IT IS!
0D50 30   03               LEAX   $03,X     SKIP OVER ADDRESS
0D52 4A                    DECA             ANO COUNT LOOKUPS
0D53 26   F7               BNE    SEEK      KEEP LOOKING
0D55 30   01       GOTIT   LEAX   1,X       SKIP TO AODRESS
0D57 EC   84               LDD    ,X        AND REAO IT
0D59 6E   8B               JMP    D,X       AND GO THERE
```

(Continued)

Color Computer has already arranged for this fetch to be diverted to $103, BKP stuffs the address of it's own service routine at $103. When all this "paperwork" is finished, BKP hands control back to CBUG, allowing you to inspect or modify anything before CBUG is instructed to jump to the routine being tested.

Note that the program execution may or may not reach the specified address. Since SWI2 is a maskable interrupt, you must be sure that the 6809 interrupt mask is cleared. Also, if your program bug bites before program execution reaches the breakpoint, then you won't learn anything except new words and another way to recover from a crash! If the breakpoint isn't reached, you must manually clear the breakpoint by replacing the code BKP grabbed, using CBUG's M command.

If the breakpoint is reached, BKP restores the code, prints the stack contents by calling CBUG's R command, then returns control to CBUG. You are then free to inspect memory and registers, trying to decide just why your "perfect" code doesn't do what you thought you told it to do. If all is well at the first breakpoint, you can continue from that location, or restart the program operation at the beginning, using a breakpoint further into the program. In the latter case, simply set a new breakpoint a few locations deeper into the program, and execute as before. If you want to follow a single piece of the action through, step by step, simply set a breakpoint for the next logical stop, and 'J' to the location of the first breakpoint. Since BKP has already replaced this code, operation proceeds as if it hadn't been stopped, unless you stopped in the middle of a time-critical segment of code. With a little diligence and care, you can locate almost any bug using this general technique.

The second program is longer, and furnishes a version of BKP that operates under control of CBUGP. It extends CBUG's abilities by jumping to a second look-up table which allows selection of a precise move routine (used to patch code), a byte search routine, and a program which prints out all single-key codes developed by the Color Computer's keyboard. (Some two-key functions are available.) Also, three "hooks" are furnished to facilitate addition of more special-purpose additions to CBUG. Finally, three subroutines, SHOW, PRTSCN, and CLRSC, can be called by your own programs to light the cursor at the current location, print a character to the screen, and

Listing 1 *(Continued)*

```
                         *COMMAND VECTORS
                         *
0D5B 42                  XTBL     FCC    'B'
0D5C 002B                         FDB    BKP-*      BREAKPOINT ROUTINE
0D5E 4B                           FCC    'K'
0D5F 013C                         FDB    KYTST-*    MAP KEYBOARD
0D61 50                           FCC    'P'
0D62 007F                         FDB    MVBLK-*    PRECISE MOVE ROUTINE
0D64 53                           FCC    'S'
0D65 00C3                         FDB    SRCHB-*    BYTE SEARCH
0D67 58                           FCC    'X'
0D68 000E                         FDB    OOPS-*     HOOK1
0D6A 59                           FCC    'Y'
0D6B 000B                         FDB    OOPS-*     HOOK2
0D6D 5A                           FCC    'Z'
0D6E 0008                         FDB    OOPS-*     HOOK3
0D70 4D                           FCC    'M'
0D71 000F                         FDB    EXITP-*    RETURN TO CBUG
               0018      TABND    EQU    *-XTBL     END OF DATA
0D73 20                           FCC    ' '
0D74 FFC0                         FDB    IN2-*      REJECT ALL OTHERS

                         *****PRINT MESSAGE*****

0D76 17    F906          OOPS     LBSR   PCRLF      RESET THE DISPLAY
0D79 30    8D 0141                LEAX   MSG1,PCR   OOPS MESSAGE
0D7D 17    F90D                   LBSR   PDATA
                         *
                         *BACK TO CBUG
                         *
0D80 37    36            EXITP    PULU   A,B,X,Y    RETRIEVE REGISTERS
0D82 35    40                     PULS   U          AND THIS ONE
0D84 16    FA09                   LBRA   MSTART+19  AND SNEAK BACK

                         *THIS ROUTINE IMPLEMENTS A BREAKPOINT ROUTINE.
                         *AFTER ENTRY, IT WILL PROMPT FOR THE BREAKPOINT
                         *ADDRESS WITH A '?', THEN RETURN TO CBUGP FOR
                         *FURTHER INSTRUCTIONS

0D87 36    36            BKP      PSHU   A,B,X,Y    SAVE REGISTERS
0D89 8E    0103                   LDX    #SWI2      GET A POINTER
0D8C 86    7E                     LDA    #$7E       JUMP OP CODE
0D8E A7    80                     STA    ,X+        AND BUILD A JUMP
0D90 CC    00C3                   LDD    #BKPFIN    TO THIS ROUTINE
0D93 ED    81                     STD    ,X++       WITH THIS VECTOR
0D95 86    3F                     LDA    #'?        SEND THE PROMPT
0D97 17    011F                   LBSR   WRT        AND WRITE IT DOWN
0D9A 86    60                     LDA    #$60       GET A SPACE CHARACTER
0D9C 17    011A                   LBSR   WRT        THEN PRINT IT
0D9F BD    0651                   JSR    BADDR      GET AN ADDRESS
0DA2 9F    18                     STX    DXBFR      SAVE THE ADDRESS
0DA4 1F    10                     TFR    X,D        MOVE IT HERE, TOO
0DA6 17    0110                   LBSR   WRT        PRINT CONTENTS OF A REGISTER
0DA9 1F    98                     TFR    B,A        SHUFFLE AND THEN
0DAB 17    010B                   LBSR   WRT        WRITE CONTENTS OF B REG
0DAE EC    9F 0018                LDD    [DXBFR]    GET THE BREAKPOINT CODE
0DB2 DD    1A                     STD    DDBFR      AND SAVE IT
0DB4 CC    103F                   LDD    #$103F     STUFF THE SWI2 OPCODE
0DB7 ED    9F 0018                STD    [DXBFR]    HERE TO APPLY THE BRAKES
0DBB 17    00C5                   LBSR   CLRSC      ERASE THE DISPLAY
0DBE 37    36                     PULU   A,B,X,Y    GET 'EM BACK
0DC0 7E    07AD                   JMP    WARMS+4    AND RETURN TO MONITOR
0DC3 1F    43            BKPFIN   TFR    S,U        SAVE HARDWARE STACK POINTER
0DC5 BD    07D9                   JSR    REG        PRINT THE STACK
0DC8 36    06                     PSHU   D          SAVE D
0DCA DC    1A                     LDD    DDBFR      RETRIEVE BREAKPOINT CODE
0DCC ED    9F 0018                STD    [DXBFR]    AND SEND IT HOME
0DD0 37    06                     PULU   D          GET D BACK
0DD2 7E    07A9                   JMP    WARMS      RETURN TO MONITOR

                         *THIS ROUTINE COMPUTES A CHECKSUM IN X; EACH BYTE
                         *SHOULD BE PASSED IN TXTCHR AND THE UPDATED CHECKSUM
                         *RETURNED IN DSBFR.  USE AS SUBROUTINE ONLY.
0DD5 36    34            CHKSM    PSHU   B,X,Y      SAVE REGISTERS
0DD7 9E    20                     LDX    DSBFR      GET CURRENT CHECKSUM
0DD9 D6    12                     LDB    TXTCHR     AND THE CURRENT BYTE
0DDB 3A                           ABX               GET THE SUM AND
0DDC 9F    20                     STX    DSBFR      THEN SAVE IT
0DDE 37    34                     PULU   B,X,Y      RESTORE REGISTERS
0DE0 39                           RTS               AND GO HOME

                         *THIS ROUTINE WILL MOVE A BLOCK OF DATA TO ANOTHER
                         *LOCATION.  PASS THE BLOCK LENGTH (BYTES) IN DSBFR,
                         *THE SOURCE ADDRESS IN DYBFR, AND THE DESTINATION
                         *ADDRESS IN DUBFR.  ALLOWANCE MADE FOR BUFFER OVERLAP.

                         *****SETUP*****

0DE1 34    76            MVBLK    PSHS   A,B,U,X,Y  SAVE REGISTERS
0DE3 17    F9E3                   LBSR   GETADR     GET TWO ADDRESSES
0DE6 9F    1E                     STX    DUBFR      END ADDRESS
0DE8 109F  1C                     STY    DYBFR      START ADDRESS
0DEB 17    F863                   LBSR   BADDR      GET # OF REPEATS
0DEE 9F    20                     STX    DSBFR      AND SAVE IT
```

Listing 1 *(Continued)* `*****DO IT!*****`

```
                       *ALLOW FOR POSSIBLE BUFFER OVERLAP

0DF0 DC      1C                LDD     DYBFR      GET START ADDRESS
0DF2 93      1E                SUBD    DUBFR      GET DISTANCE BETWEEN BLOCKS
0DF4 27      11                BEQ     EXIT2      SAME ADDRESS, WHY BOTHER?
0DF6 2D      14                BLT     REV        MOVE CODE FROM BOTTOM FIRST
                       *NOTE: THIS MOVE ALLOWS UNWANTED CODE TO BE
                       *OVERWRITTEN.  USE WITH CARE!

0DF8 9E      20                LDX     DSBFR      GET NUMBER OF BYTES TO MOVE
0DFA 109E    1C                LDY     DYBFR      ALSO START ADDRESS
0DFD DE      1E                LDU     DUBFR      AND DESTINATION START
0DFF A6      A0        B1      LDA     ,Y+        LOAD ONE BYTE
0E01 A7      C0                STA     ,U+        AND PUT IT DOWN
0E03 30      1F                LEAX    -1,X       COUNT THE OPERATIONS
0E05 26      F8                BNE     B1         LOOP UNTIL DONE
0E07 35      76        EXIT2   PULS    A,B,U,X,Y  RESTORE REGISTERS
0E09 7E      0DB0              JMP     EXITP      AND RETURN TO GO

                       *THIS MOVE ALLOWS CODE TO BE OPENED UP TO INSERT
                       *ONE OR MORE OP CODES FOR A PATCH.

0E0C DC      20        REV     LDD     DSBFR      GET NUMBER OF BYTES TO MOVE
0E0E D3      1C                ADDD    DYBFR      MOVE POINTER TO BOTTOM OF BUFFER
0E10 1F      02                TFR     D,Y        AND LOAD SOURCE POINTER
0E12 DC      20                LDD     DSBFR      GET BYTE COUNT AGAIN
0E14 D3      1E                ADDD    DUBFR      COMPUTE DESTINATION BUFFER END
0E16 1F      03                TFR     D,U        AND LOAD DESTINATION POINTER
0E18 9E      20                LDX     DSBFR      ONE MORE TIME!
0E1A 31      3F        B2      LEAY    -1,Y       POINT TO FIRST BYTE
0E1C A6      A4                LDA     0,Y        GET ONE BYTE
0E1E 33      5F                LEAU    -1,U       POINT TO NEXT TARGET
0E20 A7      C4                STA     0,U        AND SHOOT
0E22 30      1F                LEAX    -1,X       COUNT THE PASSES
0E24 26      F4                BNE     B2         LOOP UNTIL DONE
0E26 20      DF                BRA     EXIT2      THEN BLOW THE JOINT
                       *THIS ROUTINE SEARCHES FOR A SPECIFIED BYTE
                       *PASSED IN TXTCHR; PASS THE STRING LENGTH TO
                       *SEARCH IN DYBFR, AND THE BUFFER START ADDRESS
                       *IN DSBFR.  RECORD THE BYTE LOCATION IN
                       *DSBFR.  RETURN $FFFF IN DSBFR FOR TEST FAILURE.
0E28 36      36        SRCHB   PSHU    A,B,X,Y    SAVE FOUR
0E2A BD      0661              JSR     BYTE       GET CHARACTER
0E2D 97      12                STA     TXTCHR     PUT IT UP SAFELY
0E2F BD      06BD              JSR     OUTS       PRINT A SPACE
0E32 BD      07C9              JSR     GETADR     THEN GET STRING PARAMETERS
0E35 109F    20                STY     DSBFR      SAVE THE ADDRESS
0E38 9F      1C                STX     DYBFR      AND THE STRING LENGTH
0E3A 96      12                LDA     TXTCHR     CHARACTER TO FIND
0E3C 9E      20                LDX     DSBFR      AND THE START ADDRESS
0E3E 109E    1C                LDY     DYBFR      STRING LENGTH
0E41 A1      80        CS1     CMPA    ,X+        LOOK FOR IT
0E43 27      18                BEQ     CS2        GOT IT, SET POINTER
0E45 31      3F                LEAY    -1,Y       NOT IT, COUNT IT ANYWAY
0E47 26      F8                BNE     CS1        LOOP UNTIL DONE
0E49 CC      FFFF              LDD     #$-1       NOT FOUND, SET A FLAG
0E4C DD      20        SAVFLG  STD     DSBFR      PLANT THE FLAG HERE
0E4E 97      FB                STA     PARAM      SORT OUT MS BYTE
0E50 BD      06AE      SHOWIT  JSR     OUTHEX     PRINT MS BYTE OF ADDRESS
0E53 D7      FB                STB     PARAM      DO THE SAME
0E55 BD      06AE              JSR     OUTHEX     FOR THE REST
0E58 37      36        EXIT1   PULU    A,B,X,Y    RESTORE REGISTERS
0E5A 7E      0D34              JMP     IN2        AND ASK FOR MORE
0E5D 30      1F        CS2     LEAX    -1,X       POINT AT THE TARGET
0E5F 1F      10                TFR     X,D        STUFF LOCATION IN D
0E61 20      E9                BRA     SAVFLG     CLEAN UP AND LEAVE

                       *THIS ROUTINE LIGHTS THE CURSOR AND SAVES THE
                       *CHARACTER CURRENTLY POINTED TO.

0E63 36      02        SHOW    PSHU    A          SAVE A
0E65 A6      9F 0088           LDA     [CURPTR]   GET THE CHARACTER POINTED TO
0E69 97      10                STA     DPLCH      AND SAVE IT, JUST IN CASE
0E6B 8A      8F                ORA     #$8F       MAKE IT A GRAPHICS CHARACTER
0E6D A7      9F 0088           STA     [CURPTR]   SO IT WILL SHOW UP
0E71 37      02                PULU    A          GET A BACK
0E73 39                        RTS                GO BACK

                       *THIS ROUTINE WRITES TO THE DISPLAY BUFFER AND
                       *TURNS ON THE CURSOR BY A CALL TO SHOW.  PASS
                       *THE CHARACTER TO BE DISPLAYED IN PARAM.

0E74 36      36        PRTSCN  PSHU    A,B,X,Y    SAVE REGISTERS
0E76 9E      88                LDX     CURPTR     GET PRESENT CURSOR LOCATION
0E78 96      FB                LDA     PARAM      GET THE OUTPUT CHARACTER
0E7A A7      80                STA     ,X+        WRITE IT WITH ELECTRONS
0E7C 9F      88                STX     CURPTR     TELL THE NEW CURCOR LOCATION
0E7E 8D      E3                BSR     SHOW       AND TURN ON THE CURSOR
0E80 37      36                PULU    A,B,X,Y    RETRIEVE THE REGISTERS,
0E82 39                        RTS                THEN RESUME OPERATION
```

(Continued)

clear the screen, respectively. These features are available from CBUG, but not in the same form.

This extension of CBUG operates very much like CBUG, except that it is entered *via* a "J" command from CBUG, and some of the routines bounce back after one pass. Here's a brief summary:

BKP (Breakpoint routine), entered from CBUGP *via* "B": Sets up the breakpoint and then bounces back to CBUG and lets CBUG enter the routine under test. Entry form: B XXXX, where XXXX is the hex address of the breakpoint.

MVBLK: Allows the program code to be sliced down the middle to insert any number of bytes of missing code, or closed up to delete any number of bytes of superfluous code. Entry form: P XXXX YYYY ZZZZ, where XXXX is the address of the first byte to be moved, YYYY is the target location for that byte, and ZZZZ is the number of bytes to be moved. In the latter case, if you need to open up a 200-byte program to insert new code immediately after the 50th byte, (convert numbers to hex, assuming the program begins at 1000) then XXXX = 1033 and ZZZZ = 0032. If the needed patch is three byres long, then YYYY = 1035. After execution of one complete move, MVBLK returns to CBUG.

SRCHB: Searches for any single-byte value and reports the location if found, or FFFF, if not found. Enter from CBUGP using S XX YYYY ZZZZ, where XX is the value sought, YYYY is the address where to start searching, and ZZZZ is the number of bytes to search. Returns to CBUGP for further searches.

"M" command: Returns to CBUG.

CBUG — An Assembly Language Monitor for the Color Computer

One of the very first pieces of assembly language software available for the TRS-80C™ Color Computer was CBUG©, sold for $29.95 by The Micro Works, P.O. Box 1110, Del Mar CA 92014. This well-documented, assembly language monitor has a number of excellent and useful features, especially for those who wish to preserve the essential character of the Color Computer and still learn about assembly language programming on Motorola's "super" 8-bit processor, the MC6809.

MICRObits

Deadline for MICRObits: 20th of second month before publication; i.e., March 20th for May issue. Send typewritten copy with $25 per insertion. (Subscribers: first ad at $10.)

(Continued)

Listing 1 *(Continued)*

```
                        *THIS ROUTINE CLEARS THE SCREEN AND SETS THE
                        *CURSOR AT THE TOP OF THE SCREEN.

0E83 36    32           CLRSC   PSHU    A,X,Y       SAVE THREE
0E85 8E    05FF                 LDX     #SCREND     TAIL END CHARLIE
0E88 108E  0200                 LDY     ##200       NUMBER OF DISPLAY CELLS
0E8C 86    60                   LDA     #$60        SCREEN BLANK CHARACTER
0E8E A7    82           OUTSC   STA     ,-X         BLANK THE CELLS
0E90 31    3F                   LEAY    -1,Y        AND COUNT THEM
0E92 26    FA                   BNE     OUTSC       SPIN UNTIL DONE
0E94 9F    88                   STX     CURPTR      REMEMBER THE CURSOR LOCATION
0E96 8D    CB                   BSR     SNOW        LIGHT THE CURSOR
0E98 37    32                   PULU    A,X,Y       GET 'EM BACK
0E9A 39                         RTS                 DO THE NEXT MAGIC TRICK

                        *A RESEARCH PROGRAM TO MAP KEYBOARD CODES.
                        *WARNING--CALL IT ONLY FROM CBUG!!

0E9B 36    36           KYTST   PSHU    A,B,X,Y
0E9D BD    067F                 JSR     PCRLF       RESET THE DISPLAY
0EA0 BD    0618         READ    JSR     ENTRY       GET A KEY
0EA3 81    5E                   CMPA    #$5E        TEST FOR A KNOWN ONE
0EA5 27    0D                   BEQ     STOP2       AND USE IT TO ABORT
0EA7 8D    10                   BSR     WRT         OTHERWISE, OUTPUT TO SCREEN
0EA9 BD    06BD                 JSR     OUTS        PRINT A SPACE
0EAC BD    0627                 JSR     NEX         PRINT HEX VALUE OF CHARACTER
0EAF BD    067F                 JSR     PCRLF       RESET THE DISPLAY
0EB2 20    EC                   BRA     READ        THE GO GET ANOTHER KEY
0EB4 37    36           STOP2   PULU    A,B,X,Y     GET 'EM BACK
0EB6 7E    0D34                 JMP     IN2         MAMA IS CALLING!
0EB9 97    FB           WRT     STA     PARAM       SAVE THE CHARACTER
0EBB 8D    B7                   BSR     PRTSCN      AND GO PRINT IT
0EBD 39                         RTS                 RETURN TO THE BOSS

                        *****TEXT FOR OOPS MESSAGE*****

0EBE 4E 4F 54 20        MS01    FCC     'NOT IMPLEMENTED'
0EC2 49 4D 50 4C
0EC6 45 4D 45 4E
0ECA 54 45 44
0ECD 00                         FCB     0           DELIMITER

                        *****TEXT FOR PROMPT*****

0ECE 0D 0A              XPMPT   FCB     $D,$A       RESET THE DISPLAY
0ED0 43 42 55 47                FCC     'CBUG ?'    PROMPT
0ED4 50 20 3F
0ED7 20                         FCB     $20         AND A SPACE
0ED8 00                         FCB     0           DELIMITER
0ED9 0000                       FDB     0           DELIMITER
                                END
```

Listing 2

```
                      *BKP: COPYRIGHT 1981 BY RALPH TENNY
                      *REFERENCE: CBUG, COPYRIGHT 1981 BY
                      *THE MICRO WORKS, INC.

                      *****REFERENCES FROM CBUG*****

                      *NOTE: CHECK THESE LOCATIONS WITH THE VERSION OF
                      *CBUG THIS EXTENSION WILL BE USED WITH!

               0088   CURPTR  EQU   $88       CURSOR POINTER
               061E   SWRIT   EQU   $61E      ENTRY FOR OUTEY
               0651   BADDR   EQU   $651      READ BINARY ADDRESS
               077D   HSTART  EQU   $77D      HARD START ENTRY
               07A9   WARMS   EQU   $7A9      WARM START ENTRY
               07D9   REG     EQU   $7D9      PRINT STACK

                      *****EQUATES AND BUFFER DEFINITIONS FOR BKP*****

0030                          ORG   $0030
0030 0000             CODE    FDB   0         LOCATION FOR CODE AT
                                              BREAKPOINT
0032 0000             PNTR    FDB   0         LOCATION OF BREAKPOINT
               0103   SWI2    EQU   $103      VECTOR FOR SWI2

                      *****REFERENCE TO TRS80C ROM*****

               A92B   CLS     EQU   $A92B     BASIC CLS COMMAND
                      *THIS ROUTINE IMPLEMENTS A BREAKPOINT ROUTINE.
                      *AFTER ENTRY, IT WILL PROMPT FOR THE BREAKPOINT
                      *ADDRESS WITH A '?', THEN RETURN TO CBUG FOR
                      *FURTHER INSTRUCTIONS

0EE0                          ORG   $EE0
                      *ENTER HERE FROM CBUG USING 'J' COMMAND

0EE0 34    40         BKP     PSHS  U         SAVE CBUG ENVIRONMENT
0EE2 36    36                 PSHU  A,B,X,Y   SAVE REGISTERS
```

(Continued)

Keystroke	Command Description
G	Returns command to the calling program.
R	Displays register list.
M 1234	For memory examine and change, beginning at $1234.
I 1234 2345 67	Inserts $67 in memory from $1234 to $2345.
T 0123 1234 2345	Transfers block of memory from $0123 through $1234 to new location beginning at $2345.
J 1234	Jumps to user machine language subroutine at $1234.
C	Changes register list.
S 1234 2345 1357	MYFILE creates a machine language file on cassette tape, recording the code which appears between addresses $1234 and $2345; $1357 is the program entry point.
B 1200 X	Sets baud rate. X = C or P specifies the configuration of the printer port.
L	Loads hex data to memory.
$ 1234	Converts hex to decimal.
.12345	Converts decimal to hex.
P 0000	Moves display page.
U 1234 2345	Uploads. Transmits data to the screen and to the communications port.
D	Downloads. Data can he received from communications port.
!	Takes over SWI. Until this command is executed, the 6809 SWI instruction will cause undefined operation. After using this command, substitution of the code for SWI ($3F) for op-codes in a program will cause a break which returns control to CBUG.
AU	Auto mode. After the baud rate has been set and this command is entered, the computer emulates an intelligent terminal connected to a host system.
X	Terminal mode. This command causes the computer to emulate a CRT terminal.
R	Reset; causes a return to BASIC.

Listing 2 *(Continued)*

```
0EE4 8E    0103               LDX    #SWI2      GET A POINTER
0EE7 86    7E                 LDA    #$7E       JUMP OP CODE
0EE9 A7    80                 STA    ,X+        AND BUILD A JUMP
0EEB CC    0F1A               LDD    #BKPFIN    TO THIS ROUTINE
0EEE ED    81                 STD    ,X++       WITH THIS VECTOR
0EF0 86    3F                 LDA    #'?        GET THE PROMPT
0EF2 8D    38                 BSR    WRT        AND WRITE IT DOWN
0EF4 86    60                 LDA    #$60       GET A SPACE CHARACTER
0EF6 8D    34                 BSR    WRT        THEN PRINT IT
0EF8 BD    0651               JSR    BADDR      GET AN ADDRESS
0EFB 9F    32                 SIX    PNTR       SAVE THE ADDRESS
0EFD 1F    10                 TFR    X,D        MOVE IT HERE, IDD
0EFF 8D    2B                 BSR    WRT        PRINT CONTENTS OF 'A' REG
0F01 1F    98                 TFR    B,A        SHUFFLE AND THEN
0F03 8D    27                 BSR    WRT        WRITE CONTENTS OF 'B' REG
0F05 EC    9F 0032            LDD    [PNTR]     GET THE BREAKPOINT CODE
0F09 DD    30                 STD    CODE       AND SAVE IT
0F0B CC    103F               LDD    #$103F     SIUFF THE SWI2 OPCODE
0F0E ED    9F 0032            STD    [PNTR]     HERE TO APPLY THE BRAKES
0F12 BD    A928               JSR    CLS        ERASE THE DISPLAY
0F15 37    36                 PULU   A,B,X,Y    GET 'EM BACK
0F17 7E    07AD               JMP    WARMS+4    AND RETURN IO MONITOR
0F1A 1F    43         BKPFIN  TFR    S,U        SAVE HARDWARE STACK POINTER
0F1C BD    07D9               JSR    REG        PRINT THE STACK
0F1F 36    06                 PSHU   D          SAVE D
0F21 DC    30                 LDD    CODE       RETRIEVE BREAKPOINT CODE
0F23 ED    9F 0032            STD    [PNTR]     AND SEND IT HOME
0F27 37    06                 PULU   D          GEI D BACK
0F29 7E    0787               JMP    HSTART+10  REIURN TO MONITOR
0F2C 36    36         WRT     PSHU   A,B,X,Y    SAVE REGISTERS
0F2E 9E    88                 LDX    CURPTR     GET PRESENT CURSOR LOCAIION
0F30 A7    80                 STA    ,X+        PRINT IT OUT
0F32 9F    88                 STX    CURPTR     AND SAVE NEW LOCATION
0F34 37    36                 PULU   A,B,X,Y    GET 'EM BACK
0F36 39                       RIS               AND RESUME OPERAIION
0F37 0000                     FDB    0          DELIMITER
                              END
```

CBUG is available in a tape-based version, which loads at the start of the BASIC workspace ($0600), and a 2K ROM, which occupies either the Color Computer's socket for Advanced Color BASIC (addressed at $9000), or installed in a modified program pack (addressed at $C000). It is apparent that the program's flexibility of location stems from the fact that it is written entirely in position-independent code. Since the monitor is completely documented, including a well-commented source listing, a detailed study of the Owner's Manual constitutes an excellent self-taught course in programming the 6809.

The adjacent list of commands provides an idea of the capability of the monitor.

MICRO

MICRObits *(continued)*

OSI Superboard Cabinet and Accessories

Pre-cut pine cabinet kit $27.95 ppd; RS-232 interface kit $9.95 ppd; 600 baud/band kit $8.95 ppd., complete with hardware and well-illustrated instruction sets. Free catalog of software, hardware, Molex, joysticks, cassettes, kits and more.

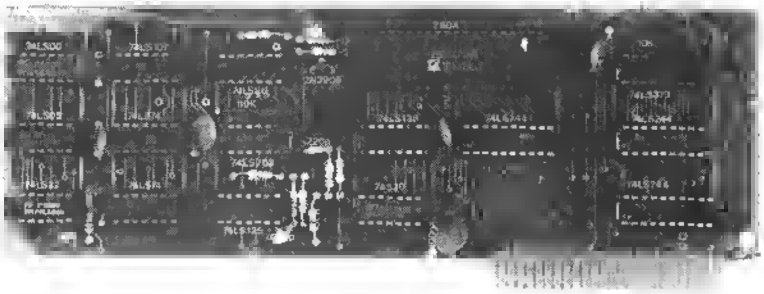
Dee Products
Department M
150 Birchwood Road
Lake Marion, IL 60110

SYM-1 Hardware and Software

Expand SYM's RAM and ROM with W7AAY boards. Learn how to add disks and build S-100 interface. Software allows use of RAE to create and edit BASIC programs and to disassemble programs and data to RAE files. SASE for details.

Blalock and Associates
P.O. Box 39356
Phoenix, AZ 85069
(602) 993-4604

(Continued on page 85)



Multiprecision Addition — A Comparison of 6809 and 6502 Programming

by Gregory Walker and Tom Whiteside

The euthors use 32-bit eddition routlnes to demonstrete severel edventeges of programming the MC6609 over the 6502. The finel routlnes ere designed to be celled es subroutines from enother progrem.

Mucb attention has focused recently on switcbing from the 6502 to the MC6809. Since the MC6809 is arcbitecturally similar (and, we believe, superior) to the 6502, the transition is both easy and worth the effort. Robert Tripp's four-part series "It's Time to Stop Dreaming" (June, July, August, September issues of MICRO) was a good overview of the similarities and differences of the 6502 and the MC6809. In this article, we will carry the description furtber with some concrete programming examples. Every attempt was made to squeeze every bit of performance out of the 6502 in tbese comparison runs. Less effort was needed for the MC6809 since its 16-bit registers and powerful instructions and addressing modes make "trickery" unnecessary (but unfortunately still possible).

A 32-bit addition subroutine was cbosen because providing multiprecision arithmetic capability is a common problem on eight-bit microcomputers. Like tbe byte-move problem, it can be solved in several different ways by trading off between code size, execution speed, and generality. We will present a 32-bit addition subroutine programmed in two different ways and see how our two processors compare.

It is often possible to trade off efficiency for generality in writing a subroutine. It is necessary to keep in mind how a subroutine will be called by tbe larger system. One major source of errors in large assembly language programs comes from destroying the contents of processor registers. We have added tbe restriction tbat tbese subroutines must leave all processor registers unchanged, but an exception is made in the case of the condition code register. Tbe condition flag register is not preserved, so that the carry flag may reflect the result of the addition.

Figure 1 shows a 6502 program that adds two 32-bit numbers togetber. The numbers and result are stored at fixed locations on the zero page. The bytes for eacb number are stored in the same order as 6502 addresses, least significant byte first. The actual addition is the fastest that can be written: eacb consecutive byte is added by a separate set of in-line instructions.

Figure 1: **6502 program to add 32-bit numbers in-line. (Time = 63 cycles.)**

SAMPLE SETUP FOR ONE OPERAND FOR 32-BIT ADD ON ZERO PAGE

```
                         (SIZ  TIM)
     ...
     LDA    OPR1           2    3    MOVE ALL FOUR BYTES OF
     STA    A1             2    3      OPERAND "OPR1" INTO
     LDA    OPR1+1         2    3      SIMULATED 32-BIT
     STA    A1+1           2    3      REGISTER "A1"
     LDA    OPR1+2         2    3
     STA    A1+2           2    3
     LDA    OPR1+3         2    3
     STA    A1+3           2    3
     ...
                          16   24    CYCLES
```

6502 ROUTINE TO ADD 32-BIT NUMBERS WITH ADDENDS IN ZERO PAGE AND ALL CODE IN-LINE

```
                         (SIZ  TIM)
ADD32 EQU   *
      PHA                  1    3    SAVE A-REGISTER
      CLC                  1    2
      LDA   A1             2    4    ADD LEAST-SIGNIFICANT
      ADC   A2             2    4      PAIR OF BYTES
      STA   RESULT         2    4
      LDA   A1+1           2    4    ADD SECOND PAIR OF
      ADC   A2+1           2    4      BYTES
      STA   RESULT+1       2    4
      LDA   A1+2           2    4    ADD THIRD PAIR OF
      ADC   A2+2           2    4      BYTES
      STA   RESULT+2       2    4
      LDA   A1+3           2    4    ADD MOST-SIGNIFICANT
      ADC   A2+3           2    4      PAIR OF BYTES
      STA   RESULT+3       2    4
      PLA                  1    4    RESTORE A-REGISTER
      RTS                  1    6
                          28   63    CYCLES
```

Figure 2 shows the corresponding MC6809 subroutine. Many of the MC6809 instructions should be familiar to 6502 programmers. In reading the MC6809 program, note that the MC6809 stores its multi-precision values in the opposite order from the 6502: the most-significant byte is stored in the lower memory address.

Figure 2: 6809 program to add 32-bit numbers in-line on the direct page. (Time = 50 cycles.)

SAMPLE OPERAND SET-UP FOR 32-BIT ADD ON DIRECT PAGE

```
        ...                    (SIZ  TIM)
        LDD     OPR1            2     5     MOVE FOUR BYTES OF
        STD     A1              2     5       "OPR1" INTO SIMULATED
        LDD     OPR1+2          2     5       32-BIT REGISTER "A1"
        STD     A1+2            2     5
        ...
                                8    20     CYCLES
```

MC6809 ROUTINE TO ADD 32-BIT NUMBERS WITH ADDENDS IN ZERO PAGE AND ALL CODE IN-LINE

```
                               (SIZ  TIM)
ADD32   EQU     *
        PSHS    D               2     7     SAVE D-ACCUMULATOR
        LDD     A1+2            2     5     ADD LEAST-SIGNIFICANT
        ADDD    A2+2            2     6       16-BIT QUANTITIES
        STD     RESULT+2        2     5
        LDD     A1              2     5     ADD MOST-SIGNIFICANT
        ADCB    A2+1            2     4       16-BIT QUANTITIES
        ADCA    A2              2     4
        STD     RESULT          2     5
        PULS    D, PC           2     9     RESTORE D AND RETURN
                               18    50     CYCLES
```

The MC6809 performs the addition in two 16-bit chunks. Two instructions are used to add the most-significant 16-bits because the MC6809 lacks a 16-bit add-with-carry instruction. The final instruction pulls the old contents of the D accumulator and the program counter from the stack, which conveniently restores the processor state and returns from the subroutine in one fell swoop.

We can see from figures 1 and 2 that the MC6809 subroutine is both smaller and faster than the 6502 version. Each single MC6809 instruction tends to require more bytes and more machine cycles than a similar 6502 instruction, but the more powerful MC6809 instruction set allows the problem to be solved with fewer instructions overall.

As we said before, a subroutine exists within a larger system. From a systems point of view, the above two subroutines suffer several problems. Most important is the use of fixed storage on the direct page. In essence, the locations labelled A1, A2, and RESULT, are simulated 32-bit registers that the 6502 and MC6809 both lack.

These addition routines operate very quickly, but a significant amount of time is needed to set up the operand values before each subroutine call. Four loads and four stores are required just to move one of the values into a simulated regirster.

A second problem is that these routines cannot easily be adapted to solve other, similar problems. A general multi-precision addition subroutine would be written using an iterative loop, so that different length operands could be handled just by changing the loop counter.

Figure 3 shows a 6502 subroutine that answers both these problems. It uses a loop to add the consecutive bytes together and it uses indirect addressing to allow operands to reside anywhere in the 6502's address space. The Y register acts as the loop counter and as an index into the multi-precision operands. It is initialized with a value of 3, which causes the loop to be executed four times. Since the operands are stored most-significant byte first, the index is a positive number which is decremented to zero. Unfortunately, this usage is not consistent with the order of address storage on the 6502. It was forced on us because the 6502 does not have an instruction that causes a branch when a negative index is incremented through zero.

This subroutine is somewhat shorter than the previous 6502 routine, but requires almost twice the execution time. The decrease in set-up time needed before calling the subroutine partially compensates for this extra time. In this case it is only necessary to initialize three 16-bit pointers on the zero page, instead of initializing three 32-bit operands.

This subroutine provides a more general solution to the problem of multi-precision arithmetic. It is easily modified to use operands of different sizes by changing the loop count. Even the calling sequence, manipulating pointers as it does, would not have to be changed for different length operands.

Figure 4 shows the corresponding MC6809 program. Here we use the MC6809's 16-bit index reigsters to hold pointers to the operands. Each byte of the operands is added in the 8-bit A accumulator, while the B accumulator serves as a loop counter and index into the operands.

Once again, the MC6809 program is smaller in size and executes faster than the equivalent 6502 program. The main advantage of the MC6809 proves to be its 16-bit-long index registers and the instructions that manipulate 16-bit data. They remove the extra memory cycles needed for indirect addressing on the 6502 and greatly simplify the programmer's task. The MC6809 handles address calculations as easily as the 6502 handles calculations with eight-bit integers.

Conclusion

In this article we have used actual programming examples to compare the 6502 and the MC6809 in solving real-world problems. The MC6809 outperforms the 6502 in this and nearly every other application.

While speed of execution and program size are always important measures, we have also tried to show ways that the MC6809 eases the task of programming. In particular, we have seen that a major limitation of the 6502 is its dependence upon zero-page addressing. As programs increase in complexity, there is an increased demand for the limited zero-page space. Complex 6502 systems such as disk operating systems and high level languages

compete heavily for zero-page locations. Bookkeeping becomes necessary to track which routines clobber which zero-page variables, and it becomes more difficult to control routine "interaction" through the zero page.

Byte efficiency and speed are reduced as it becomes necessary to reinitialize "temporaries" and to use absolute addessing. With the MC6809's 16-bit index reigsters, there is no zero-page demand for storing indirect pointers. Furthermore, the MC6809 makes storing temporary variables on the stack easy and efficient so there is less reason to use zero-page space. Finally, the MC6809 has a direct page register. Even if the zero page does clog up, it is easy to switch to another page in memory.

We beleive the MC6809 is a worthy successor to the 6502. Applications that used the 6502 will find a new vitality on the MC6809.

Acknowledgement

We want to express our thanks to Tony Fourcroy for testing the programming examples.

Tom Whiteside is a 6-year Motorolan and works with the Microprocessor design group. Gregory Walker likes to program computers, especially the MC6809. They may be contacted at Motorola, Inc., Microprocessor Design, Maildrop MZ880, 3501 Ed Bluestein Blvd., Austin, Texas 78721.

Figure 3: **6502 program to add 32-bit numbers with loop and indirection. (Time = (21*4) + 28 = 84 + 28 = 112 cycles.)**

CALLING SEQUENCE FOR 6502 INDIRECT-ADDRESSING ADDITION ROUTINE

```
                            (SIZ  TIM)
        ...
        LDA     #OPR1_L       2    2     PUT POINTER TO FIRST
        STA     A1            2    3       OPERAND INTO A1 ON
        LDA     #OPR1_H       2    2       ZERO PAGE
        STA     A1+1          2    3
        LDA     #OPR2_L       2    2     PUT POINTER TO SECOND
        STA     A2            2    3       OPERAND INTO A2 ON
        LDA     #OPR2_H       2    2       ZERO PAGE
        STA     A2+1          2    3
        LDA     #RSLT_L       2    2     PUT POINTER TO RESULT
        STA     RSLT          2    3       INTO RSLT ON ZERO PAGE
        LDA     #RSLT_H       2    2
        STA     RSLT+1        2    3
        JSR     ADD32         3    6     CALL 32-BIT ADD
                                         SUBROUTINE

                             27   36     CYCLES
```

6502 SUBROUTINE TO ADD 32-BIT NUMBERS WITH A LOOP AND POINTERS TO OPERANDS ON ZERO PAGE

```
                            (SIZ  TIM)
ADD32   EQU     *
        PHA                   1    3     SAVE A AND Y
        TYA                   1    2       REGISTERS
        PHA                   1    3
        LDY     #3            2    2     LOOP COUNT-1 AND INDEX
        CLC                   1    2       IN Y
L1      LDA     (A1), Y       2    5     LOOP: GET OPERAND BYTE
        ADC     (A2), Y       2    5       ADD OPERAND BYTE
        STA     (RESULT), Y   2    6       STORE RESULT BYTE
        DEY                   1    2       DECREMENT LOOP INDEX
        BPL     L1            3    3       LOOP UNTIL ZERO COUNT
        PLA                   1    4     RESTORE A AND Y REGISTERS
        TAY                   1    2
        PLA                   1    4
        RTS                   1    6
                             20
```

Figure 4: **6809 add of 32-bit numbers in byte-wise loop. (Time = (20*4) + 21 = 80 + 21 = 101 cycles.)**

MC6809 CALLING SEQUENCE FOR 32-BIT ADD

```
                         (SIZ  TIM)
        ...
        LEAX    A1,PCR     3    5     ADDRESS OF A1 INTO X INDEX REGISTER
        LEAY    A2,PCR     3    5     ADDRESS OF A2 INTO Y INDEX REGISTER
        LEAU    RSLT,PCR   3    5     ADDRESS OF RSLT INTO U INDEX REGISTER
        LBSR    ADD32      3    9     CALL SUBROUTINE ADD32
        ...
```

MC6809 ADD OF 32-BIT NUMBERS USING A LOOP

```
                         (SIZ  TIM)
ADD32   EQU     *
        PSHS    D          2    7     SAVE THE D ACCUMULATOR
        LDB     #3         2    2     INITIAL INDEX AND COUNT-1 IN B
        ANDCC   #$FE       2    3     CLEAR CARRY BIT
L1      LDA     B, X       2    5     LOOP: GET OPERAND BYTE(INDEXED)
        ADCA    B, Y       2    5       ADD SECOND OPERAND BYTE
        STA     B, U       2    5       STORE RESULT, INDEXED
        DECB               1    2       DECREMENT LOOP COUNT
        BGE     LI         2    3       LOOP UNTIL COUNT IS NEGATIVE
        PULS    D, PC      2    9     RESTORE D AND RETURN
                          17
```

MICRO

FLEX: An Operating System for the 6809

by Dale Puckett

FLEX is a widely supported operating system for 6800- and 6809-based microcomputers. Its history, features, and applications are discussed.

I was shocked, yet pleasantly surprised last month while reading a journal that reports only news about the SS-50 bus. There was an advertisement for a new plug-in board. Nothing unusual, but this hoard was designed to plug into an Apple. Why would any company run an ad for an Apple board in the flagsbip publication of the 68XX family?

The advertisement for the EXCEL-9 made sense when I read on. The board uses a 6809 microprocessor and has its own monitor. It lets the Apple owner switch between the 6809 or 6502 from eitber machine language or BASIC programs.

Its hardware features were interesting too — printer spooling, multitasking, 64K of RAM, programmable timers, etc. — but it was the software side of the advertisement that really caught my eye.

Software Comes First

The ad's first selling point, listed above all of the hardware details, read: "EXCEL-9 FLEX, a famous DOS, Assembler and Editor included." Later in the list it mentioned that TSC 6809 BASIC, Extended BASIC, Precompiler, Sort/Merge, etc., were coming soon. As an extra selling point, ESD Labs Co., LTD of Mission Hills, California, the hoard's manufacturer, had included the FLEX DOS in the price of the hoard.

Although the EXCEL-9 isn't the subject of this article, it's appearance spurred me to do some additional research. Looking through the ads in a recent issue of MICRO, I noticed several other pro-FLEX movements. The Computerist of Chelmsford, Massachusetts, was offering FLEXI Plus, a 6809-hased single hoard microcomputer. It, too, runs under TSC's FLEX. The same company was also advertising FOCUS, a 6809-based micro with commercial quality keyboard, dual double-sided, double-density disks with more than 640K of storage on line, and memory-mapped video featuring bit-mapped graphics, user-definable character sets, reverse video, etc. Its operating system? FLEX.

Stellation Two was offering a plug-in hoard for the Apple called The Mill. It gives you a 6809 microprocessor with multi-tasking and multi-user capability. Microware of Des Moines, Iowa, is busy installing OS-9 on this board. I believe that it will only be a matter of time before someone brings FLEX up on The Mill.

Owners of the Radio Shack Color Computer, which already sports a 6809E microprocessor, will soon be able to run the FLEX operating system. Frank Hogg Labs of Syracuse, New York, has it on the market now. This version runs on the standard Radio Shack controller so Color Computer owners can have the best of hoth worlds: fantastic color graphics made possible by Microsoft's Extended Color BASIC, and the ever-growing library of sophisticated systems and applications software written to run under the FLEX operating system.

Ability to Run on Many Machines Pays Off

All of this hardware information makes me stop and think. How can these manufacturers offer new processors and operating systems for microcomputers that have been around for several years — machines that already have their own established operating systems and hundreds of satisfied users?

In an attempt to answer that question, this article will look at FLEX from Technical Systems Consultants, Inc., (TCS) of Lafayette, Indiana, in great detail.

FLEX — Its Roots

Almost every piece of software available for the 68XX family of processors is supplied on a FLEX-formatted disk. The trend started back in 1977 with mini-FLEX, a 4K operating system that resided from \$7000 to \$7FFF on SWTPC's 6800 hox. But soon that 4K system gave way to FLEX 2.0, an 8K system which lived in high memory between \$A000 and \$BFFF. We had something going for us that no one else had — a disk operating system that would run on every 68XX machine. As a bonus, FLEX was versatile, reliahle and easy to use from a high level language like BASIC or from our own assemhly code.

Frank Hogg Lahoratory went into business during 1979 to fill the demand for high quality FLEX-based software. The firm has since become the leading international distributor of systems and applications software for the 6809.

A look at a recent ad revealed that the company handles software from the major houses, TSC and Microware, as well as several dozen programs from independent authors. Application programs include: Dataman, a random database management system; SPELL-TEST, an extremely versatile spelling checker; READTEST, a program that tests the readability of English prose; DynaStar, a cursor-based editor that is extremely easy to use; The Bill Payer System, a series of 28 programs that automate the drudgery of paying the bills; and XFORTH, an interpreter that is totally FLEX-compatible and which supports an entire family of applications software including the Oshorne General Ledger, etc.

A Closer Look

The FLEX operating system gives you a powerful set of system commands which allow you to control all disk operations directly from the terminal. Yet, at the same time, it lets the system's programmer use a smorgasbord of disk access and file management routines. And each routine is thoroughly documented.

To the casual user the Utility Command Set is probably the most important part of the FLEX system. This set of more than two dozen highly useful commands resides on a system disk. Individual commands are loaded into memory when needed. They allow you to save, load, copy, rename, delete, append or list disk files. And these simple English words are actually the commands that you type. A complete listing of the supplied utilities is shown in table 1.

There are two other major parts of the FLEX system: The File Management System and the Disk Operating System. Together they give you fully dynamic file space allocation, automatic removal of bad sectors on a disk, automatic space compression and the ability to match the system to your terminal.

Standard System Requirements

FLEX requires 8K of high memory and a minimum of 12K of low memory. The 6809 version runs at \$C000 to \$DFFF. The 6800 versions still reside at \$A000. A minimum of two disk drives is required by most utilities. Although it is possible to operate with one drive, it isn't much fun.

On the majority of the SS-50 computers, FLEX is booted into memory by a single-letter command in the monitor. In about two seconds a banner is printed and you are asked for a date. After this is entered you will see the famous FLEX prompt, "+ + +." The three plus signs mean that the operating system is ready to accept your command.

Your files are put into sectors on the disk. Each sector holds 256 bytes of information. Four of these are used to tell FLEX where to read or write its next sector, and the remaining 252 hold your data. When you delete a file, the sectors you had been using are automatically released to the system and become available for use by new files. This is known as dynamic allocation.

Your FLEX files will have filenames containing up to eight alphanumeric characters plus a three-character extension. The extension lets you and the system know what type of information is in the file. APPEND.CMD, for example, is a command which allows you to combine two files together into a third file. STARS.BAS is usually a BASIC source file which runs on one of the many BASIC interpreters available to FLEX users.

It is possible to specify the drive on which you want the system to search for a file. However, most of us use the default system, or work, drives, a FLEX convention that makes life easy. A utility command lets us change the drive assignments at any time. For example, "ASN S=0, W=1" will assign drive zero as the system drive and drive one as the work drive. Then, if "LIST THISFILE" were typed, FLEX would go to drive zero and read in the command

Table 1

Name	Function
APPEND	Append two or more files into a third file
ASN	Assign the System or Work drives
BUILD	Place a short text file on a disk
CAT	List a catalog of the files on a disk to the terminal
COPY	Copy one file to another
DATE	Print or change the system date
DELETE	Delete a file from the disk
EXEC	Use lines of text in a file as command lines
GET	Load a file from the disk into memory
I	Get the input from specified file instead of terminal
JUMP	Execute machine code at Hex address
LINK	Point boot routine to a specific file for start up
LIST	Print a text file on the terminal
MON	Return to the system monitor ROM
NEWDISK	Initialize a new disk in the proper format
O	Re-direct output to the specified file
P	Re-direct output to the printer
PRINT	Spool output from the file to the printer
PROT	Set the protection status of a file
QCHECK	Check status of file in print queue
RENAME	Change the name of a file on the disk
SAVE	Save memory to disk
TTYSET	Set terminal parameters
VERIFY	Turn verify mode on or off
VERSION	Print version of program on terminal
XOUT	Delete all files with an .OUT extension

The Utilities above are standard with FLEX. Many vendors supply additional commands which use their hardware. For example, GIMIX of Chicago, Illinois, has a command which reads the time from the clock chip on their CPU card, etc. The Utilities below come in an extra package and may be purchased from TSC.

Name	Function
CHECK	Compare two disk files and report to terminal
CMPMEM	Compare binary file on disk to memory
CONTIN	Used to repeat complex EXEC command files
DIR	Similar to CAT, but it prints all directory information
DUMP	Dump a disk file in Hex and ASCII
ECHO	Echo an ASCII string to the terminal
EXTRACT	Take specific lines from one file and put them in another
FILES	Similar to CAT, but not as detailed
FIND	Find a string of characters in a disk file
FREE	Report free space remaining on a disk
HECHO	Echo a hex character to the terminal
MAP	Print the load addresses and transfer address of a file
MEMEND	Read the FLEX MEMEND address and report or change
PDEL	A prompting delete
RUN	Load and execute a position-independent program
SPLIT	Split a text file into two new files
ZAP	Delete files in a match list without prompting

Table 2

Address	Contents
$C080-$C0FF	Line Buffer
$CC00	TTYSET Backspace Character
$CC01	TTYSET Delete Character
$CC02	TTYSET End of Line Character
$CC03	TTYSET Depth Count
$CC04	TTYSET Width Count
$CC05	TTYSET Null Count
$CC06	TTYSET Tah Character
$CC07	TTYSET Backspace Echo Character
$CC08	TTYSET Eject Count
$CC09	TTYSET Pause Control
$CC0A	TTYSET Escape Character
$CC0B	System Drive Numher
$CC0C	Working Drive Numher
$CC0E-$CC10	System Date Registers
$CC11	Last non-ASCII character
$CC12	User Command Tahle Address
$CC14-$CC15	Line Buffer Pointer
$CC16-$CC17	Escape Return Register
$CC18	Current Character
$CC19	Previous Character
$CC1A	Current Line Number
$CC1B-$CC1C	Loader Address Offset
$CC1D	Transfer Flag
$CC1E-$CC1F	Transfer Address
$CC20	Error Type
$CC21	Special I/O Flag
$CC22	Output Switch
$CC23	Input Switch
$CC24-$CC25	File Output Address
$CC26-$CC27	File Input Address
$CC28	Command Flag
$CC29	Current Output Column
$CC2B-$CC2C	Memory End
$CC2D-$CC2E	Error Name Vector
$CC2F	File Input Echo Flag
$CCC0	Printer Initialize
$CCD8	Printer Ready Check
$CCE4	Printer Output

The information above is listed to give you an idea of the magnitude of the information the FLEX programmer has available about bis operating system. The actual documentation that comes with the system gives complete details.

file LIST. It would then go to drive one and open the file THISFILE.TXT and list it on the terminal.

Redirect the Output

Now let's assume that you would like to list THISFILE on the printer instead of tbe terminal. You would simply type: P LIST THISFILE. If you wanted to build a disk file that contained a catalog of all the command files on the disk in your work drive, you would type: O CATALOG CAT.CMD. This would open the output file CATALOG.OUT and then direct tbe output of CAT to this file instead of the terminal. Later you could LIST tbe output file. Or you could PRINT it while you are working on something else. This very bandy process is known as spooling. Spooling makes it possihle to print a 45-page listing from an assembler while you edit a new source file.

Any errors you make are reported to you in English. FLEX does this hy maintaining a file of error messages on your system disk. If the file management system or DOS generates an error, the system reads the error number, finds the corresponding record on the file, and prints it on your terminal.

The FLEX Memory Map

One of the hest features of this operating system is the fact that everything is completely documented. For example, the programmer's manual lists every memory location that contains any information of interest. You can check a handy chart and know just where to PEEK to find the character used hy the system as its backspace symbol, how many columns the user has on his terminal, etc. Table 2 lists this information.

TSC has completely documented 22 individual routines that may be called hy the systems programmer. They are vectored from a jump tahle so the calls are always at the same location, even though the user's version of FLEX may he different. This feature saves you a lot of work.

For example, I frequently check SPELLTEST, my spelling checker program for FLEX systems, to see if a character is alphanumeric or not. With FLEX it is easy.

```
JSR FMS       get a character from file
JSR CLASS     alphanumeric?
BCS NONAL     it's not, go
   .
   .
   .
```

I get a cbaracter by calling the FMS. I check it hy calling a routine called CLASS. In two lines of code I have accomplished what could have taken many, if I'd had to write my own CLASS routine.

Another example comes from READTEST, my readahility tester.

```
LEAX NUMPW,PCR     point to personal
                   word count
LDB #1             tell FLEX to use
                   leading spaces
JSR OUTDEC         print the number
                   in decimal
LEAX NUMMSG,PCR    point to message
JSR PSTRNG         let FLEX print it
   .
   .
   .
```

Here, to tell the user how many personal words he used in his text, I simply pointed the 6809's X register to the location of the two-hyte (16-hit) word, set the B register not equal to zero, and called the FLEX routine OUTDEC to print it. I then pointed the X register to an English language message and called another FLEX routine to print it. Again, if I had to write a routine to output a decimal number and another to output a string of characters, it would have taken a lot more code. Tahle 3 shows the routines that are available to programmers using the FLEX operating system

The File Management System

This is the part of the system that lets your DOS talk to your disk hardware. It allocates all file space and removes it when a file is deleted.

You communicate with the FMS by using a file control block (FCB). These 320-byte blocks of RAM memory tell the FMS the name of a file, which drive it is located on, its length, etc. To talk to a disk file, you either read or write one character at a time through the FCB. Instead of calling an output routine such as the famous MIKBUG OUTEEE, you call the FMS.

```
LDA #'A          put the character in
                 A register
LEAX FCB,PCR     point X register to the
                 FCB
JSR FMS          send it out to disk
BNE ERROR        go on error
   .
   .
   .
```

The code above would send the character "A" out to a disk file. In practice it is actually a little simpler since you usually leave the X register pointing to the FCB for the duration of an output routine.

Tabla 3

Address	Function
$CD00	Coldstart address
$CD03	Warmstart address
$CD06	DOS main Loop Re-entry point
$CD09	Input Character
$CD0F	Output Character
$CD15	Get Cbaracter, honors TTYSET parameters
$CD18	Put Cbaracter, honors TTYSET parameters
$CD1B	Input into line buffer
$CD1E	Print a String
$CD21	Classify a Character: alpba or no
$CD24	Print a Carriage return/line feed
$CD27	Get Next Cbaracter from Buffer
$CD2A	Restore I/O vectors
$CD2D	Get a File specification
$CD30	Load a File
$CD33	Set an Extension code
$CD36	Add B Rcgister to X Register
$CD39	Ouput a Decimal Number
$CD3C	Output a Hex Number
$CD3F	Report an Error
$CD42	Get a Hex Number
$CD45	Output a Hex Address
$CD48	Input a Decimal Number
$CD4B	Call DOS as a subroutine
$CD4E	Check Terminal Input Status

Used in this way, the disk looks no different to your program than a computer terminal. You may have one file open for reading and another open for writing. In fact, you may have as many files as you need open at one time, as long as you have assigned a separate file control block to each one.

As a programmer, you communicate with the File Management System by using function codes. For example, the number "1" is to open a file for read. To perform this operation you need only store "1" in the first byte of the file control block, point the X register to the block, and call FMS as a subroutine. If the operation is successful, FMS will return with the carry clear. If not, the carry bit will be set and the number code of the error will be in the second byte of the FCB. You can then read that byte and see if it is something you expected, such as end-of-file. After reading this byte you can take the appropriate action. Table 4 provides a look at function codes available to FLEX programmers.

Table 4

Number Code (decimal)	Function
1	Open For Read
2	Open For Write
3	Open for Update
4	Close File
5	Rewind File
6	Open Director
7	Get Information Record
8	Put Information Record
9	Read Single Sector
10	Write Single Sector
11	Reserved
12	Delete a File
13	Rename a File
14	Reserved
15	Next Sequential Sector
16	Open System Information Record
17	Get Random Byte from Sector
18	Put Random Byte in Sector
19	Reserved
20	Find Next Drive
21	Position to Record N
22	Backup One Record

Summary

FLEX supports random files and can reach any sector in a file after no more than two disk reads. It is also easy to reach a specific character in a file by doing a small calculation using the number of bytes in a sector.

This operating system has many other features that make it a dream to program at the assembly level. But, more importantly, it is user-friendly and its syntax is simple. In fact, you'll find it much easier to use at the command level than CP/M (the popular Z-80-based operating system). When you consider this and couple it with the fact that a large base of very sophisticated application programs already runs under this operating system, it is easy to see why the hardware firms mentioned earlier made the choice to offer the FLEX operating system.

The author may be contacted at 14753 Endsley Turn, Woodbridge, Virginia 22193.

MICRO

MICRO Reviews in Brief

Product Name: **Color Scripsit**
Equip. req'd: TRS-80 Color Computer, 4K minimum; line printer
Price: $40.00
Manufacturer: Tandy Radio Shack
P.O. Box 2625
Fort Worth, TX 76113

Description: Color Scripsit is the TRS-80C version of Tandy's word processors. It consists of a text editor and formatter and is designed for the home computer market. The text editor is screen-oriented and uses the four arrow keys for cursor movement. The 23 functions include: tab control; character and word delete; block delete; move and copy; global search and change; and a routine that allows you to hyphenate words. The formatter allows lines up to 132 characters long. The screen will scroll to the right as entered, and text is viewed through the 32-character window when line lengths longer than 32 characters are selected. Pagination, headers and footers are also supported. Lines may be centered, or aligned left or right. Multiple line spacing and variable page lengths are also allowed. File storage is cassette-based.

Pluses: The program is in a ROM PAK, therefore nearly all RAM is available for text storage, 31,528 bytes in the 32K machine. Global search ignores upper/lower case differences unless otherwise specified. Keys are repeating when held down and text can be changed by just typing over the undesired text. Merging from cassette files is allowed, and ASCII files from other sources or programs in ASCII format are accepted. Text files are saved either in ASCII or in a compact form. Format standards are saved to tape with the text. Print options include single line, partial, or entire document.

Minuses: Lack of lower case display generator sometimes makes it hard to tell whether a letter is upper case or lower case. Right justification is not supported. Some keyboard characters are not available; e.g., brackets, arrows, and back slash. The right scrolling display is sometimes disconcerting, though text can be entered and formatted later in some cases. No indication of page length or number of pages is given until the document is printed.

Documentation: A well-written 8½" × 11," 40-page manual is provided, and includes many examples of text to enter and process. I located no errors in the manual, and the only part I had trouble understanding was on setting up headers and footers.

Skill level req'd: This program is for the average consumer who wants a word processor for his TRS-80C. Good quality copy can be produced with only an evening's study.

Reviewer: John Steiner

Product Name: **AIM Language ROM Switcher (ALRS)**
Equip. req'd: Rockwell AIM 65 Computer
Price: $55.00
Manufacturer: Forethought Products
87070 Dukhobor Road
Eugene, OR 97402

Description: The ALRS is a small printed circuit board which plugs into AIM 65 ROM sockets Z25 and Z26. On board the ALRS are six ROM sockets which accept 2332-type ROMs. The ROM sockets are organized as three pairs of two sockets each. Each pair occupies the address range of $B000 through $CFFF. An on-board switch (there are provisions for a remote switch) determines which ROM pair is active at any given time. As a bonus, one of three small LEDs lights to provide a visual indication of the active ROM pair.

Pluses: The ALRS is ideal for switching between Rockwell BASIC, PL/65, and FORTH. It saves a lot of wear and tear on the AIM sockets. It also minimizes damage to the language ROMs themselves from static discharge and mechanical stress.

Minuses: The top of the ALRS board is not solder-masked. Thus, the traces for the address lines are exposed. Exercise the normal precautions, especially if your AIM 65 is not enclosed, against letting specks of solder and wire clippings foul the computer.

Documentation: Three pages of documentation include installation and operating instructions, a schematic, and a parts list. Due to the nature of the product, the instructions are brief, but they are thorough and clear.

Installation: Consists of plugging the ALRS board into the AIM ROM sockets. The ALRS plugs are perfectly aligned with the AIM sockets, making this operation a snap. Once installed, a rubber foot on the bottom of the ALRS board provides the only other mechanical support needed.

Notes: The ALRS board is not designed to work with Rockwell's Pascal ROMs since Pascal is not available as a two-ROM chip set. Having six ROMs connected to sockets Z25 and Z26 will obviously consume more current than would two ROMs. Normally this should not cause a problem, but you may wish to verify that your power supply can handle the extra load anyway.

Reviewer: Christopher J. Flynn

Product Name: **Hi-res Secrets**
Equip. req'd: Apple II with Applesoft in ROM
Price: $125.00
Manufacturer: Avante-Garde Creations
P.O. Box 30160
Eugene, OR 97403
(503) 345-3043
Author: Don Fudge
Copy Protection: 2 disks, yes; 2 disks, no
Language: Applesoft, 6502 machine language with commented source provided

Description: An educational graphics utility package for the generation of hi-res shapes using novel techniques. Contains commented machine language utilities with source code and extensive teaching material.

Pluses: This four-disk package contains a 263-page book on hi-res graphics. Its purpose is to teach several novel approaches to hi-res shape creation and motion. Two of the "secrets" are the use of block graphics and Hplot shapes. Block graphics moves the binary data defining the shape around the hi-res memory, thereby providing motion to the shape. Many utilities are provided for creating block shapes from scratch, "scanning" them from already existing hi-res screens, and creating shape tables from existing

(Continued on next page)

block graphics. Hplot shapes are machine language versions of graphics normally created through the use of the Applesoft HPLOT command. These perform much more rapidly and allow for smoother action. The manual and disks contain many other secrets covering subjects such as: page flipping, sounds, font creation and 560-point resolution, color fill-in and color filtering.

Don Fudge has an objective of conveying information to Apple users. This package is not just a series of utilities; rather it is an attempt at educating on the use of hi-res graphics and related subjects. Don's sense of humor and light style make for easy reading of a fairly technical subject.

Minuses: The manual is an extensive collection of ideas which may seem overwhelming at times, especially to the less-experienced programmer. Constant references to other software packages sold by Avante-Garde detract somewhat from the presentation. Although the manual is in its third printing, the first meaty chapter, "Shapes and Other Mysteries," contained several errors. Two addresses are referred to as $3001 and $3000, which should be $3C01 and $3C00 (top of page 22). I would like to have seen a brief summary of the block shape and Hplot shape idea early in the manual just to clarify the most basic concepts used.

One of the most interesting utilities, Instant Graphics, is not well documented. While a reference card is provided, no overview of the utility is given. The manual indicates that an 88-page document can be obtained from AG.

Skill level required: introductory knowledge of machine language. Familiarity with machine language generation of graphics will permit more use of the utilities provided.

Reviewer: David R. Morganstein

Product Name:	**Grafix SEB-1 and SEB-2 Color Hi-res Graphics Boards**
Equip. req'd:	OSI SEB-1 for 1P and Superboard; SEB-2 for 48-pin bus systems
Price:	$59/$199/$239 for bare board/kit/assembled
Manufacturer:	Grafix 911 Columbia Avenue N. Bergen, NJ 07047

Description: Grafix boards use the 6847 video display generator to produce color graphics with up to 256 × 192 resolution. The highest resolution mode has only one color while lower resolution modes can have up to eight different colors. Upper case alphanumeric characters (not OSI character set) are also displayed. In addition to graphics, the SEB-1 contains 16K of 2114 type memory for program storage. The SEB-2 adds a floppy disk controller to OSI bus machines.

Pluses: Guard bands are provided, thus all dots are visible with none lost to monitor overscan. Many different modes of color graphics are available. Colors appear as shades of gray on a B&W monitor. Output can be video or R.F. Connection is made to your present machine only through 40-pin expansion port or 48-pin bus.

Minuses: The graphics memory is only 6K leaving a 2K hole in the memory map. Output cannot be combined with OSI video. A second monitor or a switch to select outputs is required. The color oscillator causes a slight herringbone pattern in the displayed picture. The many different graphics modes are really a plus, but tend to confuse the beginning programmer.

Documentation: Assembly instructions, demonstration programs, 6847 data sheets.

Skill level required: Experienced builder for kit, assembled unit plugs in.

Reviewer: Earl D. Morris

Product Name:	**Cer-Comp Co-Resident Editor/ Assembler for the Color Computer**
Equip. req'd:	TRS-80C Color Computer with 16K
Price:	$39.95
Manufacturer:	Cer-Comp 5566 Ricochet Avenue Las Vegas, NV

Description: The Cer-Comp Color Computer Editor is coupled with an assembler, runs in R/W memory, and is distributed on cassette tape in the Color Computer tape format. Besides having 21 Editor commands, it supports 12 assembler directives, six assembly options, and seven options for two- and three-pass assembly. Assembly can be to screen or printer, and it is possible to go directly from assembly to the machine code to test the program just assembled. It is compatible with either BASIC or an assembly-language monitor. It produces compatible 6809 object code from either 6809 or 6800 mnemonics, with some syntax restrictions.

Pluses: Exceptional low price, does not require Extended BASIC, excellent flexibility, short learning curve, very versatile.

Minuses: Skimpy documentation, no listing, screen format of assembly listing difficult to read, uses too much memory by not being available in ROM.

Skill level required: Normal typing skills, familiarity with standard 6809 assembly-language conventions and understanding of advanced assembler directives.

Reviewer: Ralph Tenny

Product Name:	**Epson to Color Computer Interface**
Equip. req'd:	TRS-80 and Epson MX-80 or MX-80/FT
Price:	$60.00
Manufacturer:	Texas Computer Systems Box 951 Brady, TX 76825

Description: Interfaces the TRS-80 Color Computer to the Epson MX-80 series printers. Plugs directly into a connector inside the printer, and terminates in the four-pin DIN plug that fits the TRS-80C. Operates the MX-80 at the normal TRS-80C 600 baud.

Pluses: Provides an easily installed connection between the computer and peripheral. Eliminates the need for special serial-to-parallel interface hardware. Allows the sending of all special control codes to format the printer *via* CHR$ commands.

Minuses: The Epson graphics set is not accessible, even when using the PTFX system tape, or the 1.1 ROM. TCS is working on this to verify if it is printer- or interface-related.

Documentation: None provided, though the Epson manual provides all instructions necessary for installation.

Skill level required: Simple installation, if you are not afraid of opening electronic equipment.

Reviewer: John Steiner

MICRO

TASC.™ The Applesoft* Compiler. It turns your Apple into a power tool.

Step up to speed. TASC, the Applesoft Compiler, converts a standard Applesoft BASIC program into super-fast machine code. By increasing program execution speed up to 20 times, Microsoft gives you a power tool for Applesoft BASIC programming.

Highest capacity available. TASC will compile and run larger programs than any other Applesoft Compiler. As a disk-based system, it doesn't require the simultaneous presence of compiler and program in memory. The memory you save allows you to compile significantly bigger programs.

Power without bulk. Code expansion of up to 100% severely restricts other compilers. TASC's special code compression schemes typically limit code expansion to only 25%. You'll really appreciate that with complex programs or programs that utilize Apple's hi-res graphic pages.

More BASIC power. TASC's powerful new commands increase Applesoft BASIC programming capability. Chain with COMMON allows compiled programs to share variables, so a main menu supports several programs in a single runtime environment.

TASC's True Integer Arithmetic and Integer FOR...NEXT capabilities maximize the execution speed of compiled programs. TASC's near total compatibility with Applesoft speeds compilation of existing programs with little or no modification.

What about mistakes? You perfect your programs interactively with Applesoft. If something does slip by, TASC recovers from errors discovered in compilation and traps all runtime errors. It even permits graceful interruptions during compilation.

See for yourself. Ask for a demonstration of TASC at your Microsoft dealer. Discover the software package that turns your Apple into a power tool.

MICROSOFT
CONSUMER PRODUCTS

A Division of Microsoft Inc.
10700 Northup Way • Bellevue, WA 98004

*Applesoft is a trademark of Apple Computer, Inc.

MICRO

PET Vet

By Loren Wright

With this issue of MICRO centering on the 6809, it seems appropriate to cover the 6809 aspect of the SuperPET in a little more depth.

The SuperPET is a new computer from Commodore, aimed especially at the educational market. Included with the computer are interpreted versions of APL, BASIC, Pascal, and FORTRAN. COBOL, and compiled versions of some of these languages, are on the way (at extra cost, of course!). Also included is a serial interface, which allows files to be sent from the SuperPET to a mainframe with the same interpreter.

SuperPET Architecture

The SuperPET looks just like an 8032 from the outside, but on the inside there are a few differences. Two circuit hoards are stacked on top of the main hoard. The lower one includes the 6502 (moved up from the main board), a 6809, and the circuitry for the serial (RS-232C) interface. The upper board contains 64K of additional RAM. This may be write-protected under either switch or program control. There is also a switch to determine on power-up whether the machine is under 6502, 6809, or program control.

Since neither the 6502 nor the 6809 can address more than 64K, the extra 64K of RAM is divided into 16 hanks of 4K, and a mechanism called "hank-switching" is used to put one hank at a time into the $9000 block of the address space.

The 6809 has its own processor-dependent set of ROMs, just like the 6502 has its PET-BASIC ROMs. The rest of the SuperPET is shared — available directly to the current processor. Address ranges $A000-$E7FF and $F000-$FFFF are processor-dependent, while everything else, including the bank-switched RAM, is available to either processor.

When the SuperPET is running one of its interpreters, the 6809 is in control, the interpreter is stored in the bank-switched RAM, and the user's program is stored in the lower 32K of RAM. Most people will use the SuperPET in this configuration. However, it is possihle to use the SuperPET as an 8032, running Wordcraft, OZZ, and other business software. These programs do not utilize the extra RAM, however, and it is unlikely that future versions of these programs will, either. The business market is supported by the 8096.

You can also write your own 6809 programs using the included Assembler/Linker package. The rest of this discussion covers the features and use of this powerful package.

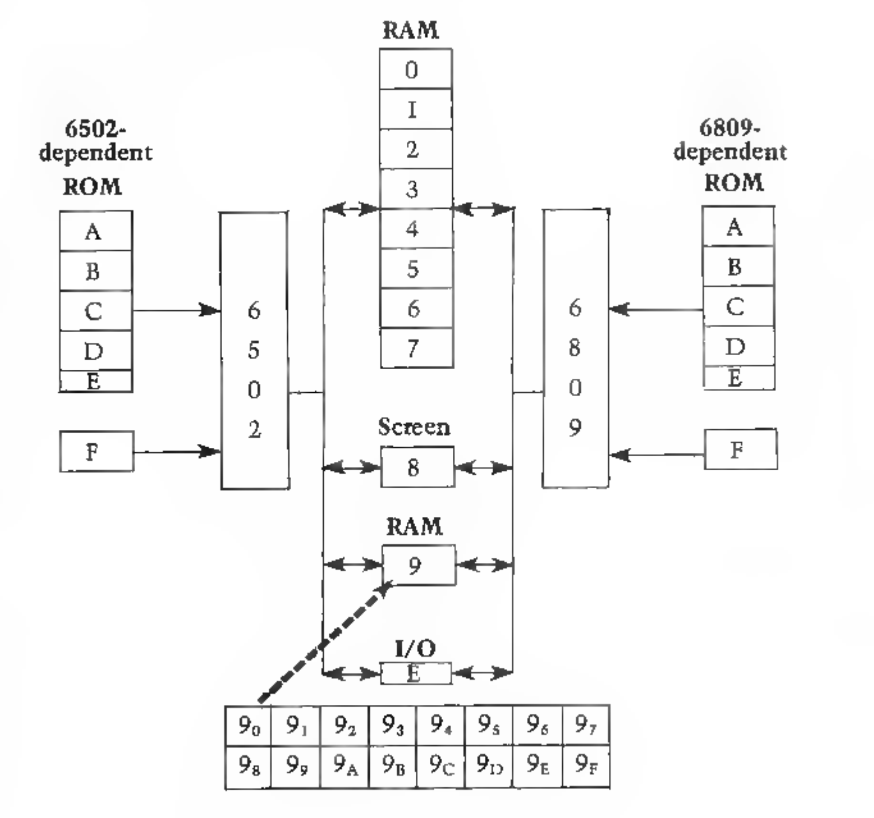
The Waterloo Assembler/Linker

When the SuperPET is powered up in the 6809 mode, one of the choices offered from the menu is "development." When this option is selected, another menu is presented with the options: asm, edit, linker, monitor, and quit. Quit returns you to the main menu.

Editor

The first step is to create a source file in the editor. This is the same editor used by Pascal, FORTRAN, and BASIC. It is hasically a powerful line editor, but PET-type screen editing and a numher of window commands are offered. Tah stops can be set to help provide a suitably indented, structured listing.

Figure 1: **SuperPET Architecture**



PET Vet *(Continued)*

Assembler

The assembler creates two files: a list file, with the object code appearing adjacent to the source, and the object file. Because the 6809 code is generally relocatable, the assembler does not require an ORG statement. The locations of the resulting object files are determined in the linking process.

There are several "structured" constructs available with the assembler: IF...ELSE...ENDIF, GUESS...ADMIT... ENDGUESS, LOOP...ENDLOOP, LOOP...UNTIL, and QUIF (which may be used within the other constructs). The condition tested by IF or QUIF may be any of the conditions tested by the 6809's branch or long-branch instructions.

The assembler also offers conditional assembly, macro capabilities, and a variety of pseudo-ops. Operands may include Boolean expressions, as well as addition, subtraction, multiplication, and division.

Linker

The linker receives instructions from a command file created with the editor. The command file includes the program origin, the names of the object modules, the names of any library files, and the name to be applied to the executable module. Bank switching, bank sizing, and names of global variables are also specified in the command file.

Monitor

To run your program, you must enter the monitor and load the module created by the linker. In addition to the usual dump, save, go, load, and register commands, there are bank, fill, passthrough, and translate (=disassembler). Additional commands set and clear breakpoints for debugging.

Documentation

Like the other elements of the Waterloo "micro-" software, the assembler/linker is supported with a reference manual. The first part consists of a series of exercises that serve more to familiarize you with the features of the package than to teach 6809 assembly language. The remainder of the manual is a good reference on the various components of the package and the programs included in the system library.

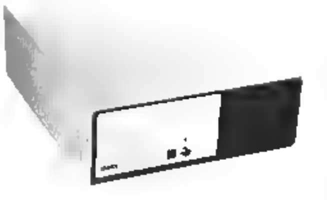
Donald Cowan of Waterloo University has written a text on 6809 assembly language programming. This text is available from WATFAC Publications Ltd., P.O. Box 803, Waterloo, Ontario, Canada N2J 4C2 for $10 (prepaid only). Some dealers may also have this book available. The next edition will be a bound book, while the first two editions are intended to be put into a three-ring binder. It is an excellent text for learning 6809 assembly language on the SuperPET.

New 8096 Software

Most business software packages available for the 8032 have now been rewritten for the 8096. In addition, new products are being produced, like the "Silicon Office" from the creators of OZZ. The package includes a versatile data base manager that allows transfer of data from one data base to another or to the built-in word processor. There is also a communications module, which allows communication between "Silicon Office" installations. My brief encounter with this software left me truly impressed. However, a package of this magnitude obviously requires a much more thorough evaluation. If "Silicon Office" (or at least its concept) is any indication of the future support we can expect for the 8096, then we will be seeing some truly fantastic software.

MICRO

GIMIX 2MHZ 6809 SYSTEMS



GIMIX offers you a variety of system packages including systems that feature BOTH **MICROWARE's OS-9** Level 1™ operating system and **TECHNICAL SYSTEMS CONSULTANTS' FLEX™**. Switch between these two predominant 6809 Disk Operating Systems, under software control, without the need to change PROMS, switches, or system configuration. System packages are also available for **MICROWARE's OS-9** Level 2 and **TECHNICAL SYSTEMS CONSULTANTS'** UniFLEX™. You can select one of our featured systems or select from our wide variety of system components to build a custom system to suit your needs.

All systems include any required CPU Board options and are completely configured to your specifications. They do not include disk drives or terminals. See pages 4 and 5 for information on 5¼" drives for installation in the CLASSY CHASSIS and/or 8" disk drives and cabinets. Any combination of 5¼" and 8" floppy disk drives, up to four drives total, can be used with systems that include controller (except UniFLEX™ systems which require 8" drives)

For information and pricing on additional options see the appropriate pages of this brochure or contact the factory.

56KB 2MHZ 6809 SYSTEMS WITH GMXBUG/FLEX/OS-9 SOFTWARE SELECTABLE

INCLUDES: CLASSY CHASSIS, 6809 PLUS CPU Board, 56K Byte STATIC RAM, #43 Two Port Serial I/O board w/cables, and...
with #58 single density disk controller (System #59) $2988.59
with #68 DMA double density disk controller (System #49) $3248.49
To substitute Non-Volatile CMOS RAM with battery back-up, add .... $ 150.00

128KB 2Mhz 6809 DMA Systems for use with TSC's UNIFLEX or MICROWARE's OS-9 Level 2

INCLUDES: CLASSY CHASSIS, 6809 PLUS CPU Board, #68 DMA Disk Controller, Two 64K Byte STATIC RAM Boards, #43 Two Port Serial I/O board w/cables, (software not included. UniFLEX™ requires 8" disk drives) $3798.39
To substitute 128KB of Non-Volatile CMOS RAM w/Battery Back-Up add .... $ 300.00

56 KB SYSTEM #29 This system can be used as the basis for a custom system to suit your special needs. It includes: CLASSY CHASSIS, 6809 PLUS CPU, 56KB STATIC RAM Board, and #43 TWO PORT SERIAL I/O board w/cables. You can add to this your choice of Disk Controllers, Memory, I/O, Software, etc. $2498.29
50 Hz version of above add .... $ 30.00

The GIMIX CLASSY CHASSIS™

6800 / 6809 SS-50 BUS MAINFRAME

The CLASSY CHASSIS includes:

A HEAVYWEIGHT, ALUMINUM CABINET (18" wide x 21" deep x 7" high) painted in a putty colored, durable baked enamel finish. The cabinet holds our 6800/6809 mother board, CV Ferro-resonant power supply, and has provisions for mounting one or two 5¼" Floppy or Winchester disk drives. The back panel is punched for 15 "D" type data connectors (25 pin) and has provisions for two removable connector plates that are available in a variety of connector configurations. Cabinets are normally supplied with two blank plates unless other types are required or specified. The cabinet includes a fan and ventilation slots which direct cooling air over the boards and power supply. The front panel has a 3 position, key locking, power switch that permits the reset switch to be locked out, preventing accidental system reset, and a three position RESET/ABORT switch. Optional filler plates are available for systems that do not use the 5¼" drive openings

The 6800 / 6809 SS-50 / C MOTHERBOARD includes:

This highly versatile motherboard is easily reconfigured for a variety of 6800 and 6809, SS-50 and SS-50C bus configurations.

GOLD PLATED connectors are used throughout to insure long lasting electrical contact and protection against corrosion.

It has fifteen 50 pin slots, 8 DIP-switch addressable 30 pin I/O slots, and a special 10 pin slot for the baud rate generator board. The fully buffered I/O block can be configured for 4, 8, or 16 decoded addresses per slot, and is DIP-switch addressable to any 32, 64, or 128 byte boundary. Extended address decoding (SS-50C) allows the I/O block to be addressed anywhere in the 1M byte address space.

The baud rate generator board provides 11 standard (16X) baud rates, from 75 to 38.4K, in 2 groups. Programming jumpers allow easy selection of up to five baud rates. The five baud rate lines on the 50 pin bus are easily disconnected from the 30 pin bus for use with SS-50C extended addressing or as user defined lines. A slow I/O circuit, for the 6809 CPU, can be used to generate an MRDY signal whenever an I/O slot is accessed (This allows, for example, using PIO Disk Controllers with a 2MHz. 6809 CPU).

All data, address, and control lines are fully terminated and separated by noise reducing ground lines on the bottom of the board.

The .090" thick, double sided P.C. board has a full ground plane Faraday Shield on the top side to further reduce noise.

The CV Ferro-resonant Power Supply features a custom designed for **GIMIX** to GIMIX specs Constant Voltage, Ferro-resonant, faraday shielded, transformer that provides brown-out and overvoltage protection and permits the system to operate properly, even under adverse AC power input conditions. It also includes an AC line filter and AC resonant capacitor, 3 DC filter capacitors, and **GIMIX** unique filter assembly board that has a clamping terminal block for easy wiring connectors. The power supply provides +8 Volts at 30 Amps, +16 Volts at 5 Amps, and −16 Volts at 5 Amps, enough to power a fully loaded system plus the two 5¼" Disk drives, including Winchester types, that can be installed in the cabinet. All supply outputs are filtered and individually fused. The standard version operates over an AC input range of 90 to 140 Volts, 60 Hz. Export versions are available for inputs of 95 to 130 or 190 to 260 volts, 50 Hz.

CABINET, MOTHERBOARD, and POWER SUPPLY assembled, burned in, and tested $1198.19
50 Hz Export versions (specify voltage) Add .... $ 30.00

Please see page 7 for information on optional front panel filler plates, disk regulator boards, back panel connector plates, and back panel cable sets.

NOTE: Due to weight restrictions GIMIX MAINFRAMES with 5" drives installed and GIMIX 8" DISK CABINETS with drives installed cannot be shipped via UPS. At the customers option we will ship these systems via UPS with the drives packed separately or via air freight (EMERY) collect, with the drives installed. Please specify the desired shipping method when ordering. Regardless of the shipping method chosen, all systems are assembled and tested as complete units before shipping.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account #73-32033. Visa or Master Charge also accepted.

Be sure to add **$30.00** for each 50Hz power supply where needed

GIMIX Inc. reserves the right to change pricing and product specifications at any time without further notice — **GIMIX**™ and **GHOST**™ are registered trademarks of GIMIX Inc. ©1982 GIMIX Inc.

GIMIX Inc. — 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

SOFTWARE AVAILABLE FOR GIMIX DISK SYSTEMS

GIMIX VERSIONS OF TSC's 6809 FLEX operating systems are available for all three **GIMIX** disk controllers. They fully support all the features of each controller and are software compatible with other versions of FLEX. **GIMIX FLEX** includes a disk FORMAT program that allows the user to pick the number of tracks to format, single or double sided disks, and where appropriate single or double density. It also supports both single (48 TPI) and double (96 TPI) track 5¼" drives and allows 80 track (96 TPI) drives to read, write, and format 40 track (48 TPI) disks. FLEX is single user and limited to 56KB systems.
Specify controller and type of drive: 8"; or 5¼" 40 or 80 track **$90.00**
NOTE: *FLEX requires a system monitor (e.g. GMXBUG or S-BUG E). When used with a SWTP CPU and S-BUG E and the GIMIX #68 DMA CONTROLLER, the GIMIX BOOTSTRAP PROM is also required.*

GMXBUG 09 includes advanced debugging capabilities as well as utility and memory manipulation routines. The standard terminal based version can be upgraded to video based for use with the GIMIX 80 x 24 Video board by changing the bootstrap PROM to the Video/bootstrap Prom. It can be used with either GIMIX DAT or SWTP DAT, but they are not required.
Price includes PROMs, Manual, and Source listing (Specify DAT) **$98.65**
Video/bootstrap or Bootstrap PROM only (included w/GMXBUG) **$30.00**

GIMIX' versions of **MICROWARE's OS-9 Level 1** are available for all GIMIX disk controllers. OS-9 includes PROMS and Disk. Microware's OS-9 Debugger is also included. Level 1 is multi-user, but limits user to 56KB Specify controller and type of drive: 8"; or 5¼" 40 or 80 track **$195.00**

★ *SYSTEM SPECIAL* ★ GIMIX offers you GMXBUG/FLEX/OS-9 selectable under software control. See System prices elsewhere in this brochure.

UNIFLEX is available for **GIMIX** Systems using the GIMIX 6809 CPU board and the #68 DMA Controller with 8" drives. It requires a minimum of 128KB of RAM. A signed license agreement with TSC is required before shipping. The SWTP DAT parts must be installed on the GIMIX CPU.
UNIFLEX . . . . **$550.00** **GIMIX boot PROM for UNIFLEX** **$50.00**

MICROWARE's OS-9 Level 2 requires a minimum of 128KB of RAM. The GIMIX DAT parts must be installed on the GIMIX CPU. GIMIX versions of Level 2 also include the Debugger *(To be available soon)* **$495.00**

A WIDE VARIETY OF LANGUAGES AND OTHER SOFTWARE IS AVAILABLE FOR THESE 6809 DISK OPERATING SYSTEMS

FOR MICROWARE's OS-9 LEVEL 1 & 2:

Macro Text Editor	$125.00	CIS COBOL	$895.00	OS-9 PASCAL	$400.00
OS-9 Assembler	125.00	Forms 2 Option	200.00	OS-9 C Compiler (Available Soon)	400.00
BASIC09	195.00				

FOR TSC's FLEX

6809 Native-Code Pascal Compiler	$200.00	Sort/Merge	$ 75.00	Standard Basic Precompiler	$ 50.00
Basic	75.00	6809 Debug Package	75.00	Extended Basic Precompiler	50.00
Extended Basic	100.00	6809 Diagnostics Package	75.00	6809 FLEX Utilities	75.00
Text Processing System	75.00	6809 Assembler	50.00	68000 Cross Assembler	250.00
Text Editing System	50.00				

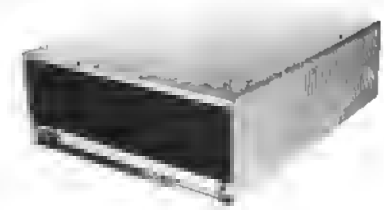
FOR UNIFLEX

1 Year Maintenace Included on all Uniflex Prices.

UniFLEX Operating System (6809)	$550.00	UniFLEX Sort/Merge	$150.00	Fortran 77 (requires relocating assembler)	$350.00
UniFLEX Basic	200.00	UniFLEX Pascal	300.00	6809 Relocating Assembler & Linking Loader	175.00
UniFLEX Basic Precompiler	150.00	UniFLEX 68000 Cross Assembler	300.00	Fortran & Relocating Assembler (pkg. deal)	450.00
UniFLEX Text Processor	150.00	Enhanced Printer Spooler	150.00		

C Compiler (Requires relocating assembler, available soon) . . . . 400.00 C Compiler & Relocating Assembler 500.00

The above software is from **MICROWARE and TSC**. Numerous offerings of languages (e.g. C, PASCAL, FORTH), utilities (e.g. spelling dictionaries, cross assemblers, disassemblers) and application packages (e.g. word processing, data base management, accounting), are available from many other software houses.



8" DISK CABINET and POWER SUPPLY. The cabinet features the same quality, styling, and finish as the GIMIX MAINFRAME and mounts two standard size 8" floppy and/or winchester disk drives. It will also hold 4 thinline 8" floppys or a combination of 2 thinline floppys and an 8" winchester.

To provide an easy means of controlling the power to an entire system from one switch, three accessory outlets, one for the computer and two for peripherals (terminals, printer, etc.), are provided. The back panel mounted power switch selects either OFF, ON, or the AUTO mode. In the AUTO mode, the power supply and two of the accessory outlets are controlled by the computer (or other device), connected to the third accessory outlet. When the computer is turned on or off, the cabinet senses the presence or absence of current flow to the computer and turns itself and the other accessory outlets on or off. Circuitry is also provided to turn AC drive motors ON and OFF under computer control. A built in fan with a washable air filter provides cooling for the power supply and drives. The back panel is punched for 4 connectors (two 50 and two 20 pin) for connections between the cabinet and the computer.

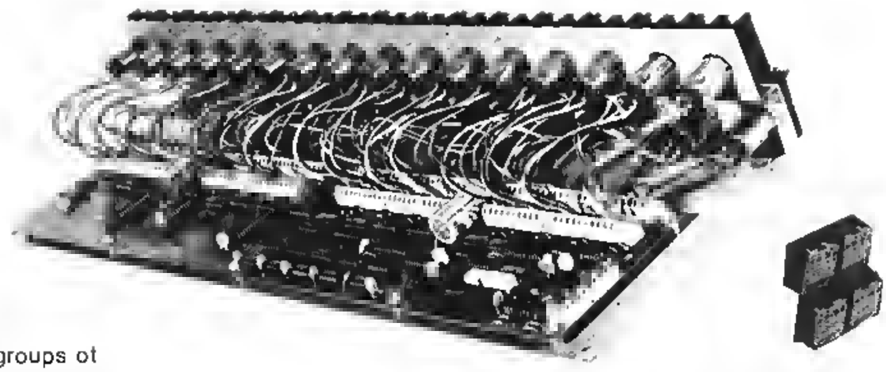
The power supply uses a constant voltage Ferro-resonant transformer for reliability and protection against brownouts and power line noise. It provides +5 Volts at 6 Amps, +24 Volts at 6 Amps, and −5 Volts at 750 Ma. continuously; with ample surge capacity for drives that require higher starting currents. The supply has two separate 24 V. outputs that can be sequenced to delay starting of the second drive until the first is up to speed.

All units are fully assembled, burned in, and tested.

8" DUAL DRIVE DISK SYSTEM: includes two double sided 8" disk drives, cabinet, power supply, and all necessary cables to connect to a GIMIX MAINFRAME or controller (see shipping notes on page 8) **$2698.88**
8" DISK CABINET ONLY: includes power supply and AC & DC power cables **Note:** Because different drive models require different AC & DC connectors, be sure to specify the quantity and model number of the drives being used when ordering. **$848.18**
For 50 Hz Export power supply, add . . . . **$ 30.00**
DRIVE CABLE: for 8" floppy drives includes connectors for the disk drives and a back panel connector for the 8" disk cabinet.
with 2 drive connectors **$44.82**
with 4 drive connectors **$67.84**
MAINFRAME CABLE: for use with the above cable; to connect the disk cabinet to GIMIX MAINFRAMES and disk controllers . . . **$45.81**
8" FILLER PLATE: used when only one drive is installed **$14.83**

Control 31 Separate AC Circuits (20 amps max. ea.)

RELAY DRIVER BOARDS FOR A.C. POWER CONTROL



4 Boards (124 relays) can be connected to one 20 ma. current loop. Each board controls 31 G.E. RR8 relays.

Use multiple serial ports for additional groups of 124 relays.

SIMPLE TO CONNECT Only *two pairs* of wires coming from your computer are needed for each set of four Realy Driver Boards, these wires may be the standard telephone type.

REMOTELY LOCATABLE. Relay Driver Boards can be conveniently located for A.C. power distribution — away from the computer and other Relay Driver Boards. The board operates in either the active or the report mode, as specified by the computer. In the active mode, the board interprets the 8-bit data received as a command to turn on or off a particular relay. Following abrief interval to allow the selected realy to operate, the board senses that relay's status (on or off). If the status is other than expected, the computer takes appropriate action, as determined by the program. A command received in the report mode has the same results, except for relay activation. This allows the mode to check relay status at any time.

If the on-board UART detects a transmission error, such as in framing, parity, or overrun, no relays are activated and no status scan occurs.

Clamping terminal blocks for wiring simple SPST-N.O. momentary contact remote switches to individual relays or groups of relays, both on and off, provide manual control as in a normal low voltage switching system, even without the computer. In event of power failures, the relays will remain in the same state that they were in when power is restored. DATA rates up to 1200 baud, allow operating up to 120 relays per second on each port.

COMPACT — Only 24" x 5"

Distances and operation of boards and relays are dependent upon wire length and gauge, and type of transformer.

RELAY DRIVER BOARD ACCESSORIES

MOUNTING BRACKET ★ custom designed to hold a Relay Driver Board and 31 relays. The bracket (26" x 8¼" x 4") and transformer will fit in a standard electrical cabinet (extra room needed for wiring) creating a neat and easily installed system.

TRANSFORMER ★ 2 Amp., 24 volts. Custom manufactured to our specs for powering a Relay Driver Board and 31 G.E. RR8 relays.

G.E. RR8 RELAYS ★ 24 volt, split coil, mechanical latching type. Once ON they stay ON (drawing no current) until they are powered OFF, and vice-versa. Each relay can handle 20 AMPS for switching lights, motors, machinery, etc. up to 277 V.A.C. — UL listed.

PRICES

RELAY DRIVER BOARD ONLY **$488.86**
BRACKET .... **$ 38.21**
TRANSFORMER .... **$ 14.24**
RELAY DRIVER PACKAGE **$1083.08**

(Relay Driver Board, 31 RR-8 Relays, Bracket and Transformer)

OPTO-BOARD FOR REMOTE SENSING $348.85

Links any computer to 34 Outside-World Signals safely
Inputs isolated to 1500 volts
Perfect for detecting closure of switches and relays
Built-In Debouncing.
Signals may range from 5 to 24 volts D.C.
Can detect signals sent by devices such as wall switches, hidden floor switches, electric eyes, alarms, smoke detector, thermostats, and a multiplicity of other applications.

All switch ports are constantly scanned by an on-board circuit. No processor time is required. A built-in memory buffer saves up to 64 closed-switch signals, permitting the processor to complete lengthy tasks between interruptions.
FULL HANDSHAKING LOGIC:
DATA READY output DATA ACCEPTED input
BUFFER FULL output RESET input
ALL OUTPUTS ARE BUFFERED AND TTL COMPATIBLE

PARTS AND CABLE SETS FOR GIMIX BOARDS AND SYSTEMS

BAUD Rate Generator Board $88.93
GIMIX double disk regulator with two 4 amp regulators to provide power for 5¼" drives 68.22
Filler plates (when no 5" drives are used), 2 required 14.92
Missing Cycle Detector 38.23
8" Disk Cable and Back Panel Connector Set 29.25
8" Disk Cable Set 44.26
5" Disk Cable Set $34.96
I/O Cable Set, each (specify board) 22.95
GIMIX 2" D Ring Binder 9.00
GIMIX 3" D Ring Binder 12.00
OPTIONAL Back Panel Connector Plates for Mainframe Choice of: Blank; SO-239; BNC; 20 & 50 Pin Header; 34 & 40 & 50 Pin Header. Connectors not included...... 8.60

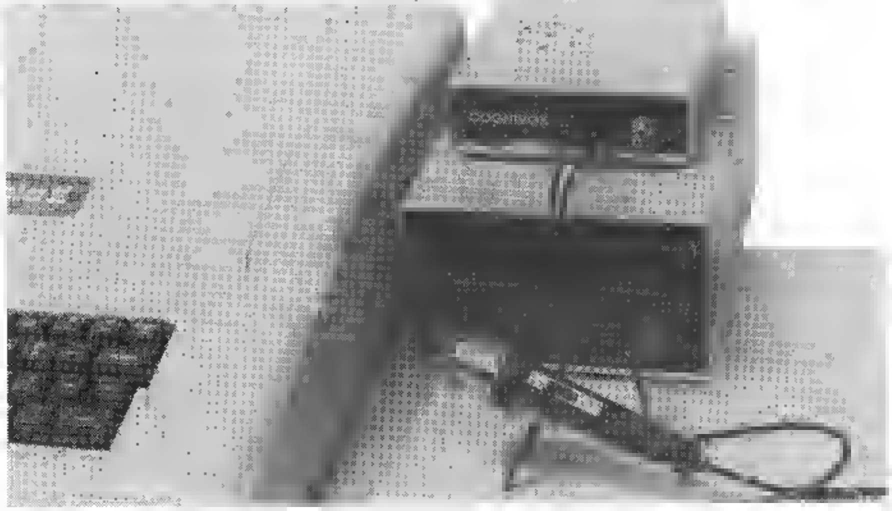
GIMIX 50 PIN PROTOTYPING BOARD

- Double sided with plated thru holes and gridded power and ground lines.
- 16 rows of pads on.100 x .300 centers; up to 72 fourteen pin ICs.
- Accepts standard 6, 8, 14, 16, 20, 24, 28, and 40 pin DIP devices.
- The entire top edge has pads for .100 x .100 header (ribbon) connectors.
- Pads for solder connections or .100 center headers on all 50 bus lines.
- Accepts 4 TO-220 regulators; 2 on the +8V & 1 ea. on the +/− 16 V lines.
- Provisions for decoupling caps distributed throughout the array
- Can be used with wire wrap, wiring pencil, solder wiring, etc.

With gold bus connectors and heat sinks — unassembled $56.66

GIMIX Inc. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX 910-221-4055

IF YOU CONSIDER THE PRICE, COGNIVOX AT $249 IS THE BEST BUY IN VOICE I/O.
IF YOU CONSIDER THE PERFORMANCE, YOU WILL BUY IT. BECAUSE COGNIVOX OUTPERFORMS ALL VOICE INPUT OR OUTPUT PERIPHERALS FOR THE APPLE II. AT ANY PRICE.



COGNIVOX VIO-1003 is a state-of-the-art Speech Recognition and voice output peripheral for the APPLE II computer. It enables the APPLE to recognize words or short phrases spoken by the user and it can talk with natural sounding voice.

SPEECH RECOGNITION

COGNIVOX recognizes words (such as "one," "enter," etc.) or short phrases (like "total amount," "net weight," etc.) from a vocabulary of 32 entries. The vocabulary entries are chosen by the user to suit his application. Then COGNIVOX is "trained" to the vocabulary by repeating each entry three times into the microphone under the prompting of the system.

During training, COGNIVOX analyzes the voice of the user and compresses all the important information in each entry into 48 bytes of data called the reference pattern. When training is complete, words spoken in the microphone are similarly analyzed and the resulting 48-bit pattern is compared with all the reference patterns to obtain a best match.

The power of COGNIVOX is derived from proprietary pattern generation and pattern matching algorithms that allow quick and easy training and give a recognition accuracy equal to much more expensive units.

Vocabularies larger than 32 words are possible by swapping reference patterns in memory using a key word, for example, "change vocabulary." Or the swap can be performed under program control.

VOICE OUTPUT

COGNIVOX can talk with a vocabulary of 32 words or short phrases. No restrictions are placed on the vocabulary which can be programmed simply by saying the words into the microphone. The speech waveform is then digitized using a data compression method and stored in memory.

When voice output is desired, the selected word or phrase is reconstructed and played back using a built-in speaker/amplifier. A jack is also provided that allows connection to external amplifiers or speaker.

This method of voice output offers two very important advantages: First, the user has full control over the selection of the vocabulary and the type and tone of voice. Second, the voice output is naturally sounding human speech which is pleasant and easy to understand. These features are not available in most other voice output devices in the market.

The voice output and speech recognition vocabularies are independent of each other and can be different. Thus it is possible to establish a dialog with the computer.

USING COGNIVOX

COGNIVOX is designed for extreme ease of use. It is a complete system, fully assembled and tested, including hardware in an instrument case, microphone, power supply, cassette with software and user manual. It plugs into the game I/O port in the APPLE and does not use up the valuable peripheral slots.

Software provided with COGNIVOX include demonstration programs and two voice operated, talking video games. All programs are unprotected so that the user can examine and modify them.

An optional diskette for DOS 3.3 includes all cassette software plus disk facilities to store and retrieve vocabularies on disk.

Adding voice I/O to your own programs is very simple. A statement in BASIC is all that is needed to either recognize or say a word. Complete instructions on how to add voice to your programs are given in the manual.

APPLICATIONS

COGNIVOX adds a whole new dimension to man-computer interaction. It can be used for data and command entry when hands and/or eyes are busy. As an educational tool. As an aid to handicapped. As sound effects generator. As a telephone answering machine. As a talking calculator, or talking clock.

The list is endless. With a BSR home controller interface it can be used to control by voice appliances and lights around the house. With an IEEE 488 interface card it can be used to control by voice instruments, plotters, test systems. And all these devices could talk back, saying their readings, alarm conditions, even their name. Finally, COGNIVOX is a super toy, a fascinating device to play with. Imagine an adventure game that talks to warn you of danger and listens to your commands!

SPECIFICATIONS

Recognizer type:
Isolated word, speaker dependent.
Vocabulary size:
32 words or short phrases for both recognition and voice response.
Dialog capability:
Recognition and response vocabularies can be different.
Word Duration
Greater than 150 ms and less than 3 seconds.
Silence gap between words:
150 ms minimum.
Training required:
Must pronounce vocabulary 3 times to train recognizer. Allows words to be individually retrained.
Recognition accuracy:
Up to 98%. Recognition accuracy depends on speaker experience and choice of vocabulary.
Type of voice output:
Digital recording of user voice.
Audio output:
130 mW
Frequency response:
100 to 3200 Hz.
Power consumption:
120 mW during recognition, 350 mW maximum during speech output.
Power supply:
9V DC, 300 mA, unregulated.
Dimensions:
5"x 6"x 1.25"
Memory requirements:
Approx. 4K bytes for program and tables. 1.5K bytes per sec. of speech for storage of voice response vocabulary (Approx. 700 bytes per word).

ORDER COGNIVOX TODAY
To order COGNIVOX by phone, call us at (805) 685-1854, 9AM to 4PM PST, Monday through Friday, and charge it to your MASTERCARD or VISA. Or you can order by mail. Send us a check or money order for $249 plus $5 shipping and handling (CA residents add 6% tax). Software on diskette (DOS-3.3) order part # DSK-3.3, $19. Foreign orders welcome, add 10% for air mail shipping and handling. COGNIVOX is backed by a 120 day warranty against manufacturing defects.

VOICETEK
Dept G, P.O. Box 388
Goleta, CA 93116

ALSO AVAILABLE for COMMODORE COMPUTERS and the AIM-65. CALL or WRITE for MORE INFO.

7SEG: PET Giant Character Set

by John Girard

Use this routine to display alpha-numeric characters in a large, seven-segment display on the screen.

7SEG
requires:
40-column PET/CBM
With slight modifications for screen size, it will run on an 80-column CBM or a 22-column VIC.

Changing the size of PET characters is impossible without major modification to your PET. But, with the print utility 7SEG, you can construct giant, seven-segment style characters on CBM/PET screens. 7SEG characters are visible up to 40 feet away and are ideal for any application where visibility is critical.

This article presents an all-BASIC core program which can be adapted to your specific needs. Some of the potential applications include clocks, counters, device status, and instrument readouts, such as digital multimeter displays.

7SEG constructs numbers by sequentially drawing the contents of seven strings, A1$ through A7$ (see figure 1). Each string prints one segment, composed of a series of spaces and cursor controls. The segments are turned on or off by adding reverse field controls to the print statements. To print an 8, for example, you would call the subroutine (program line 315):

```
PRINT"  "A1$A2$A3$A4$A5$A6$A7$;
:RETURN
```

In this example all seven strings print in reverse field. To print a 0, you would call another routine (line 235):

```
PRINT"  "A1$A2$A3$A4$A5$A6$"  "
A7$; :RETURN
```

Figure 1: **Illustration of the seven numeric segment print strings.**

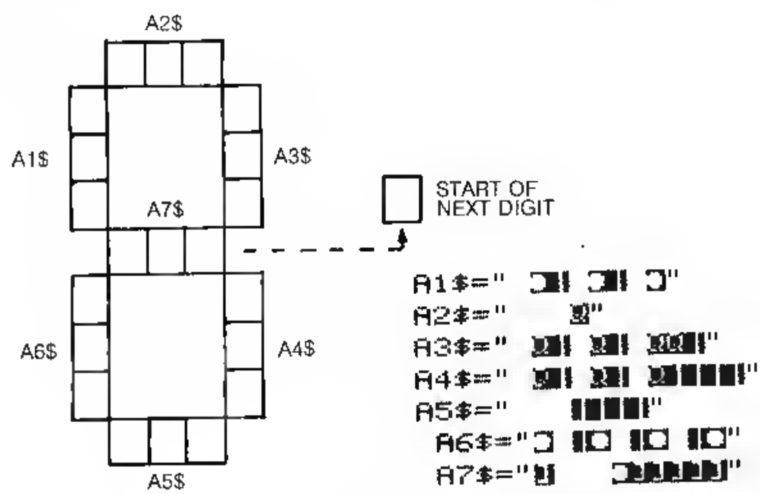


Figure 2: **Display of a floating point number.**

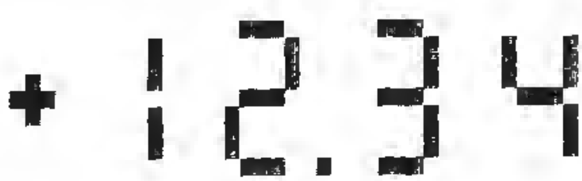
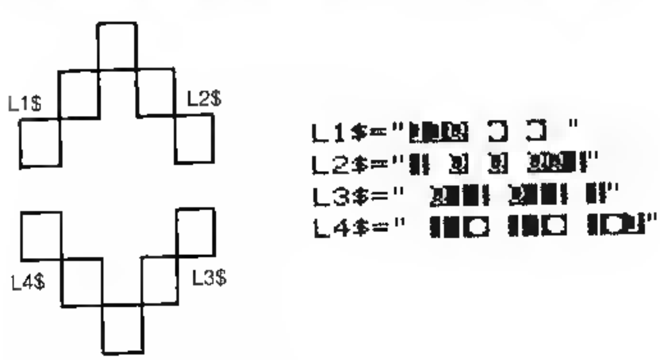


Figure 3: **Illustration of the four diagonal segment print strings.**



In the latter example, the final string (A7$) is printed off. Referring again to figure 1, note that the printing of A7$ positions the cursor at the starting point for the next character. Therefore, to print a series of numbers, you just move the cursor to the desired starting position, then simply print A1$ through A7$ over and over with the appropriate reverse field controls. This relative positioning technique allows for fast operation. A four-digit real number with floating decimal will appear in less than 1/3 second (see figure 2). To produce alphabetic displays, you will need just four more strings to handle the diagonals, L1$ through L4$, illustrated in figure 3.

Load the driver core and enter several numbers less than 10,000. Note the editing features. Leading and trailing zeroes are blanked for numbers with less than four significant digits. Enter the word "MAD" and watch the diagonals come into play. Enter the word "ERROR" and see the largest word I have squeezed into a 40-column screen. Think of the possibilities on an 80-column screen!

To illustrate the flexibility of 7SEG, I have included a short overlay program to reduce the character size and allow display of five significant digits. Simply type the overlay onto the original program.

John Girard (along with Loren Wright, MICRO's PET Vet) developed more than two dozen college-level physics programs for the University of California at Berkeley. Girard is now working as an accounting applications programmer at Pacific Telephone. His address is 676 Alma # 202, Oakland, CA 94610.

***Listing 1:* Four-digit "core" program.**

```
100 REM*** 7SEG DRIVER CORE WITH 4 GIANT SIZE DIGITS & LETTERS ***
105 REM WRITTEN BY JOHN GIRARD 11/1/80
110 REM DO NOT RENUMBER THIS PROGRAM!!
115 GOSUB 465:REM INTITIALIZE STRINGS-
120 PRINT"♫"
125 PRINT"▒■          ": INPUT"▒";A$: FL = 0
130 N = ABS(VAL(A$)): SN = SGN(VAL(A$))
135 N$ = STR$(N)
140 REM MAIN DISPLAY LOOP-------------
145 IF A$ = "MAD"   THEN 415
150 IF A$ = "ERROR" THEN 385
155 PRINT"▒■■■■■"DN$;: L = LEN(N$): IF L > 6 THEN L = 6
160 REM CLEAR DISPLAY OH ZERO RESULT
165 IF VAL(A$) = 0 THEN FOR I=1TO4:GOSUB 330:NEXT I:OLN$="":OS = 0:GOTO 125
170 REM FLOATING DECIMAL--------------
175 FOR I=1TO5:POKE DC+I*9,32
180 IF MID$(N$,I+1,1) = "." THEN FL = I:POKE DC+I*9,160
185 ON VAL(MID$(N$,I+1,1)) GOSUB 240,250,260,270,280,290,300,310,320
190 IF MID$(N$,I+1,1) = "0" THEN GOSUB 230
```

(Continued)

Listing 1 *(Continued)*

```
195 REM BLANK TRAILING ZEROS----------
200 IF I > (L-1) THEN GOSUB 330
205 IF (FL=0) AND (I=4) THEN I=5
210 NEXT I:OLN$ = N$
215 GOSUB 350:REM POLARITY-----------
220 GOTO125
225 REM SUBROUTINES BEGIN HERE--------
230 REM PRINT 7SEG 0 ----------------
235 PRINT"■"A1$A2$A3$A4$A5$A6$"■"A7$;:RETURN
240 REM PRINT 7SEG 1 ----------------
245 PRINT"■"A1$A2$"■"A3$A4$"■"A5$A6$A7$;:RETURN
250 REM PRINT 7SEG 2 ----------------
255 PRINT"■"A1$"■"A2$A3$"■"A4$"■"A5$A6$A7$;:RETURN
260 REM PRINT 7SEG 3 ----------------
265 PRINT"■"A1$"■"A2$A3$A4$A5$"■"A6$"■"A7$;:RETURN
270 REM PRINT 7SEG 4 ----------------
275 PRINT"■"A1$"■"A2$"■"A3$A4$"■"A5$A6$"■"A7$;:RETURN
280 REM PRINT 7SEG 5 ----------------
285 PRINT"■"A1$A2$"■"A3$"■"A4$A5$"■"A6$"■"A7$;:RETURN
290 REM PRINT 7SEG 6 ----------------
295 PRINT"■"A1$"■"A2$A3$"■"A4$A5$A6$A7$;:RETURN
300 REM PRINT 7SEG 7 ----------------
305 PRINT"■"A1$"■"A2$A3$A4$"■"A5$A6$A7$;:RETURN
310 REM PRINT 7SEG 8 ----------------
315 PRINT"■"A1$A2$A3$A4$A5$A6$A7$;:RETURN
320 REM PRINT 7SEG 9 ----------------
325 PRINT"■"A1$A2$A3$A4$"■"A5$A6$"■"A7$;:RETURN
330 REM PRINT A BLANK ---------------
335 IF (I>5) AND (FL=I) THEN RETURN
340 IF VAL(A$) = 0 AND (I>5) THEN RETURN
345 PRINT"■"A1$A2$A3$A4$A5$A6$A7$;:RETURN
350 REM POLARITY --------------------
355 IF OS = SN THEN RETURN
360 IF SN = 1  THEN 370
365 PRINT"■"DN$"■    ":PRINT"■    ":PRINT"■    ":GOTO 380
370 PRINT"■"DN$"■ ■ ■ ":PRINT"■    ":PRINT"■ ■ ■ ":GOTO 380
375 PRINT"■"DN$"■■    ":PRINT"    ":PRINT"    "
380 OS = SN:RETURN
385 REM PRINT "ERROR" ---------------
390 PRINT"■■"DN$;
395 PRINT"■"A1$A2$"■"A3$A4$"■"A5$A6$A7$"■■"A1$A2$A3$A4$A5$"■"A6$A7$"■■"A1$;
400 PRINTA2$A3$A4$A5$"■"A6$A7$"■■"A1$A2$A3$"■"A4$A5$A6$A7$"■■■■"A6$A7$;
405 FOR I = 1 TO 600:NEXT I
410 GOTO 120:REM RESTART LOOP
415 REM PRINT "MAD" -----------------
420 PRINT"■■■■■■■"DN$;:GOSUB 435:GOSUB 445:PRINT"■■";:GOSUB 455
425 FOR I = 1 TO 300: NEXT I
430 GOTO 120:REM RESTART LOOP
435 REM PRINT M ---------------------
440 PRINT"■"A6$"■"A1$"■■"L2$"■■■■"L1$A3$A4$"■■■■■■■■■■";:RETURN
445 REM PRINT A ---------------------
450 PRINT"■"L1$L2$A4$"■"A5$"■"A6$A7$;:RETURN
455 REM PRINT D ---------------------
460 PRINT"■"A1$"■■■"L2$L3$"■■■"A6$"■"A7$;:RETURN
465 REM LOAD STRINGS N THINGS -------
470 A1$ = " ■■ ■■ ■"
475 A2$ = "    ■"
480 A3$ = " ■■ ■■ ■■■"
485 A4$ = " ■■ ■■ ■■■■■"
490 A5$ = "    ■■■■"
495 A6$ = "■ ■■ ■■ ■■"
500 A7$ = "■    ■■■■■"
505 L1$ = "■■■ ■ ■ "
510 L2$ = "■ ■ ■ ■■■"
515 L3$ = " ■■■ ■■■ ■"
520 L4$ = " ■■■ ■■■ ■"
525 DC   = 33601
530 DN$ = "■■■■■■■■■■■■■■■■":RETURN
```

***Listing 2*: Five-digit overlay. Enter listing 1, followed by this patch.**

```
100 REM*** 7SEG PATCH FOR 5 DIGIT NUMBERS***
105 REM WRITTEN BY JOHN GIRARD 11/1/80
110 REM TO USE, OVERLAY THESE LINES ON THE CORE PROGRAM
155 PRINT"■■■■■■"DM$;  L = LEN(N$)  IF L > 7 THEN L = 7
165 IF VAL(A$) = 0 THEN FOR I=1TO5:GOSUB 330:NEXT I OLN$="" OS = 0:GOTO 125
175 FOR I=1TO6:POKE DC+I*8,32
180 IF MID$(N$,I+I,I) = "." THEN FL = 1:POKE DC+I*8,160
205 IF (FL=0) AND (1=5) THEN I=6
335 IF (I>6) AND (FL=1) THEN RETURN
340 IF VAL(A$) = 0 AND (I>6) THEN RETURN
400 PRINTA2$A3$A4$A5$"■"A6$A7$"■■"A1$A2$A3$"■"A4$A5$A6$A7$"■■■■"A6$A7$;
440 PRINT"■"A6$"■"A1$"■■"L2$"■■■■"L1$A3$A4$"■■■■■■■■■";:RETURN
450 PRINT"■■"L1$"■■"L2$A4$"■"A5$"■"A6$A7$;:RETURN
470 A1$ = " ■■ ■"
475 A2$ = "   ■"
480 A3$ = " ■■ ■■■"
485 A4$ = " ■■ ■■■■"
490 A5$ = "   ■■■■"
495 A6$ = "■ ■■ ■■"
500 A7$ = "■   ■■■■■"
```

MICRO

MICRObits *(continued)*

Apple/Morse Decoder

Apple software decodes Morse from any CW receiver. No hardware interface required. Locks onto any speed or pitch. Receive news, sports, weather and chitchat. On disk or cassette with complete documentation — $12.50.

Bob Wiseman
118 St. Andrews
Cincinnati, OH 45245

OSI Screen Editor

Never retype a line with Edit-All. Dynamic insert/delete corrects any line on the screen. With user-definable scroll window and full cursor control and screen clear. For OS-65D systems — $19.95. *Universe* (by Galaxian author) plays like *Cobra Copter* — $14.95 tape/disk.

DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

BASIC LOAD & SAVE w/filename

Tired of LOAD-SAVE times, tape documentation? Now SAVE with filenames on your cassette programs. LOAD by filename. Gives directory of tape as you go. Employs a token I/O system to decrease tape I/O by 50%! $10.95 (plus postage).

Computer Science Engineering
c/o David Sandberg
71 Reed Street
Lexington, MA 02173

OSI C1P ELIZA

Talk *with* your computer! ELIZA converses with you in plain English. It's responses are often uncanny, almost human! Cassette 8K — $9.95 ppd. Adventures, other programs available. Free list. Programmers: stop feeling rejected! Send us your programs, documentation for big profits.

Cygnus Software
781 W. Oakland Park Blvd.
Suite 432
Ft. Lauderdale, FL 33311

(Continued on page 88)

Applesoft Variable Dump

by Philippe Francois

This hendy debugging utility presents e "dump" of current verlable arrey values for Applesoft In ROM.

Applesoft Arrey Dump
requires:
Apple with Applesoft in ROM

This program follows and completes the "VARIABLE DUMP" program by Scott D. Schram from the May 1981 MICRO. The original program printed all non-subscripted variables.

My version retains the Schram skeleton program but is a little more complex since array storage is more complicated than simple variable storage.

To load "ARRAY DUMP" enter monitor mode and type machine code into memory beginning at $4000. Then save the routine to disk with "BSAVE ARRAY DUMP, A$4000,L$1A3."

To use the program load ARRAY DUMP into memory with a "BLOAD ARRAY DUMP" followed by "CALL 16384." (You may instead BRUN ARRAY DUMP.) As in the "VARIABLE DUMP" program, hit any key to stop or start the listing.

Please direct correspondence to the author at CNRS/Laboratoire D'Informatique pour les Sciences de l'Homme, 31 Chemin Joseph Aiguier, B.P. 71, 13277 Marseille Cedex 9, France.

Sample Run

```
10  DIM AA%(1,2),BB$(2),CC(3)
20 AA%(0,0) = 1:AA%(1,2) = 19999
30 CC(1) = 999.99
40 BB$(0) = "THIS":BB$(1) = "IS A":BB$(2) = "TEST"

]RUN

]BRUN ARRAY DUMP

AA%(0,0)=1
    (1,0)=0
    (0,1)=0
    (1,1)=0
    (0,2)=0
    (1,2)=19999

BB$(0)=THIS
    (1)=IS A
    (2)=TEST

CC (0)=0
    (1)=999.99
    (2)=0
    (3)=0

]D$(1)="THAT'S ALL"

]CALL 16384


AA%(0,0)=1
    (1,0)=0
    (0,1)=0
    (1,1)=0
    (0,2)=0
    (1,2)=19999

BB$(0)=THIS
    (1)=IS A
    (2)=TEST

CC (0)=0
    (1)=999.99
    (2)=0
    (3)=0

D $(0)=
    (1)=THAT'S ALL
    (2)=
    (3)=
    (4)=
    (5)=
    (6)=
    (7)=
    (8)=
    (9)=
    (10)=
```

MICRObits *(continued)*


```
                ; ************************************
                ; ***                              ***
                ; ***          ARRAY DUMP          ***
                ; ***                              ***
                ; *** ROUTINE TO DUMP ALL ARRAY    ***
                ; ***   VARIABLES TO CURRENT       ***
                ; ***    OUTPUT DEVICE             ***
                ; ***                              ***
                ; ***   BY PNILIPPE FRANCOIS       ***
                ; ***                              ***
                ; ************************************
                ;
                ;
                ; EQUATES...
                ;
                VARL   EPZ $6B         ; APSOFT'S POINTER
                VARH   EPZ $6C         ; TO ARRAY VARIABLE STORAGE
                ENDSTL EPZ $6D         ; APSOFT'S POIHTER
                ENDSTH EPZ $6E         ; TO END OF STORAGE
                CH     EPZ $24
                XSAV   EPZ $46         ; SAVE AREA FOR X REG.
                ;
                POIHTL EPZ $00         ; POINTER TO
                POINTN EPZ POINTL+1    ; CURRENT VARIABLE
                SPL    EPZ POINTN+1    ; STRING PRINT POIHTER
                SPH    EPZ SPL+1
                LEN    EPZ SPH+1       ; LENGTH OF STRING TO PRIHT
                ADDL   EPZ LEN+1       ;OFFSET POINTER TO
                ADDH   EPZ ADDL+1      ; NEXT VARIABLE
                TYPE   EPZ $D0         ; TYPE OF VARIABLE
                TYPOUT EPZ TYPE-1      ; SYMBOL TABLE OF VARIABLES
                ENDVAR EPZ TYPOUT+6    ; END OF CURRENT ARRAY VARIABLE
                SNIFT  EPZ ENDVAR+2
                INDX   EPZ SHIFT+1     ; MAX. VALUE FOR X REG.
                INDY   EPZ INDX+1      ; MAX VALUE FOR Y REG.
                NBDIM  EPZ IHDY+1      ; NUMBER OF DIMEHSIOH
                DIM    EPZ NBDIM+1     ; SIZE DIMENSION TABLE
                ;
                STROBE EQU $C010       ; KEYBOARD STROBE
                KBOARD EQU $C000       ; KEYBOARD
                ;
                ;         APPLEQUATES...
                ;     (SEE APPLE PEELED VOL2)
                ;
                GIVAYF EQU $E2F2       ; APSOFT'S INTERNAL HUMBER
                PTRFAC EQU $ED2E       ; HANDLING ROUTINES
                MOVEFM EQU $EAF9
                OUTDO  EQU $DB5C       ; PRINT CHAR IN A REG.
                CRDO   EQU $DAFB       ; PRINT A CARRIAGE RETURH
                OUTSPC EQU $DB57       ; PRINT A SPACE
                PRBLK  EQU $F948       ; PRINT THREE SPACES
                APSOFT EQU $D43C       ; APSOFT'S WARM START
                ;
                ;
                       ORG $4000

                ;
                ; DETERMINE TYPOUT'S TABLE
                ;
                START:
4000 A9A5              LDA #"%"        ; IHTEGER ARRAY (SYMBOL %)
4002 A202              LDX #$02        ; USE TWO BYTES FOR EACH ELEMENT
4004 95CF              STA TYPOUT,X
4006 A9A4              LDA #"$"        ; STRING ARRAY (SYMBOL $)
4008 A203              LDX #$03        ; USE THREE BYTES FOR EACH ELEMENT
400A 95CF              STA TYPOUT,X
400C A9A0              LDA #" "        ; REAL ARRAY (SYMBOL " ")
400E A205              LDX #$05        ; USE FIVE BYTES FOR EACH ELEMEHT
4010 95CF              STA TYPOUT,X
4012            ;
4012 20FBDA            JSR CRDO        ; PRINT A C.R.
4015 A56B              LDA VARL        ; MOVE BYTES
4017 8500              STA POIHTL      ; FROM APSOFT'S
4019 A56C              LDA VARH        ; POINTERS TO
401B 8501              STA POINTH      ; PROGRAM'S POIHTERS
401D            ;
401D A500       LOOP   LDA POINTL      ; SEE IF WE
401F C56D              CMP ENDSTL      ; ARE AT END
4021 D009              BNE PRINT1      ; NO
4023 A501              LDA POINTH      ; CHECK HIGH BYTE
4025 C56E              CMP ENDSTH      ; IF BOTN ARE EQUAL NO MORE
4027 D003              BNE PRINT1      ; ARRAY VARIABLE LEFT
4029 4C3CD4            JMP APSOFT      ; RETURN TO BASIC
402C            ;
402C            ; DETERMINE THE TYPE OF THE NEXT ARRAY VARIABLE AND
402C            ; NOTE IT 1H THE VARIABLE 'TYPE'
```

(Continued on next page)

Array Dump *(continued)*

```
402C          ;
402C          PRINT1:
402C A000           LDY #$00        ; NEXT CHAR WILL BE PRINT
402E 8424           STY CH          ; AT THE LEFT MARGIN OF THE WINDOW
4030 20FBDA         JSR CRDO        ; PRINT A C.R.
4033 B100           LDA (POINTL),Y  ; THE HIGH ORDER BIT OF THE
4035 3013           BMI TYP1        ; DETERMINE THE TYPE'S VARIABLE
4037 C8             INY
4038 B100           LDA (POINTL),Y
403A 1007           BPL TYP2
403C A903           LDA #$03        ; IT'S A STRING
403E 85D0           STA TYPE
4040 4C4E40         JMP LABEL5
4043          TYP2:
4043 A905           LDA #$05        ; IT'S A REAL
4045 85D0           STA TYPE
4047 4C4E40         JMP LABEL5
404A          TYP1:
404A A902           LDA #$02        ; IT'S AN INTEGER
404C 85D0           STA TYPE
404E          ;
404E          ; PRINT THE ARRAY VARIABLE'S NAME
404E          ; FOLLOWED BY ITS SYMBOL AND
404E          ; COMPUTE THE ADRESS OF THE NEXT VARIABLE
404E          ;
404E          LABEL5:
404E 203341         JSR PRINTN      ; PRINT VARIABLE'S NAME
4051 A6D0           LDX TYPE        ; CHOOSE IN TYPOUT TABLE THE SYMBOL

4053 B5CF           LDA TYPOUT,X    ; CORRESPONDING TO THE TYPE
4055 205CDB         JSR OUTDO       ; AND PRINT IT
4058 C8             INY
4059 B100           LDA (POINTL),Y
405B 8505           STA ADDL
405D C8             INY
405E B100           LDA (POINTL),Y
4060 8506           STA ADDH
4062 18             CLC
4063 A505           LDA ADDL
4065 6500           ADC POINTL
4067 85D5           STA ENDVAR
4069 A506           LDA ADDH
406B 6501           ADC POINTH
406D 85D6           STA ENDVAR+1
406F C8             INY             ; DETERMINE THE NUMBER
4070 B100           LDA (POINTL),Y; OF DIMENSION
4072 85DA           STA NBDIM
4074 0A             ASL             ; INDY IS THE MAX. VALUE OF Y
4075 85D9           STA INDY
4077 0A             ASL             ; IHDX IS THE MAX. VALUE OF X
4078 85D8           STA INDX
407A 18             CLC             ; SHIFT IS THE VALUE TO BE
407B A5D9           LDA INDY        ; ADDED TO THE POINTL POINTER TO
407D 6905           ADC #$05        ; ATTEMPT THE FIRST ARRAY VALUE
407F 85D7           STA SHIFT
4081          ;
4081          ; MOVE BYTES FROM SIZES DIMENSIONS OF ARRAY INTO
4081          ; DIM TABLE
4081          ;
4081 A200           LDX #$00
4083          LABEL8:
4083 C8             INY
4084 B100           LDA (POINTL),Y
4086 95DB           STA DIM,X
4088 C8             INY
4089 E8             INX
408A B100           LDA (POIHTL),Y
408C 95DB           STA DIM,X
408E E8             INX
408F E4D9           CPX INDY
4091 D0F0           BNE LABEL8
4093          ;
4093          ; INITIALISE INDEXES I,J,K.. TO ZERO
4093          ;
4093 A6D8           LDX INDX
4095 A900           LDA #$00
4097 95DA     LABEL9 STA DIM-1,X
4099 CA             DEX
409A E4D9           CPX INDY
409C D0F9           BNE LABEL9
409E          ;
409E          ; COMPUTE ADRESS OF THE CURRENT ARRAY'S FIRST VALUE
409E          ;
409E 18             CLC
409F A500           LDA POIHTL
40A1 65D7           ADC SHIFT
40A3 8500           STA POINTL
```

(Continued on next page)

MICRO

Microbes and Updates

R.L. Morris from Lynchburg, VA, offers this update to "Recursive Use of GOSUB in Microsoft BASIC" (43:68):

I enjoyed reading the above article and decided to experiment with my AIM 65 for the program calculating N-factorial. I had some problems due to the differences in BASIC between these two computers. Below is a listing that does run on the AIM 65. You can see from the "RUN" printout that the AIM quits at N=22 with an "OM" error. The only changes made from the original are in line 100 and the addition of line 170.

```
LIST
 10 INPUT "ENTER N";N
 20 IF N< 0 or INT(N) < N THEN 60
 30 GOSUB 100
 40 PRINT N; "FACTOR=";F
 50 GOTO 10
 60 PRINT N; "FACTOR UNDEFINED"
 70 GOTO 10
100 IF N=1 OR N=0 THEN 170
110 N=N-1
120 GOSUB 100
130 N=N+1
140 F=F*N
150 RETURN
170 F=1 :RETURN

RUN
ENTER N? 0
 0 FACTOR= 1
ENTER N? 1
 1 FACTOR= 1
ENTER N? 2
 2 FACTOR= 2
ENTER N? 3
 3 FACTOR= 6
ENTER N? 4
 4 FACTOR= 24
ENTER N? 21
 21 FACTOR= 5.10909422E+19
ENTER N? 22
?OM ERROR IN 100

RUN
ENTER N? -1
 -1 FACTOR UNDEFINED
ENTER N? 5
 5 FACTOR UNDEFINED
```

(Continued on next page)

Array Dump *(continued)*

```
40A5 A501              LDA POINTH
40A7 6900              ADC #$00
40A9 8501              STA POINTH
40AB           ;
40AB           ; PRINT CURRENT INDICES (I,J,K..)
40AB           ;
40AB           LABEL2:
40AB A928              LDA #"("
40AD 205CDB            JSR OUTDO
40B0 A6D8              LDX INDX
40B2           LABEL1:
40B2 B5DA              LDA DIM-1,X
40B4 A8                TAY
40B5 CA                DEX
40B6 8646              STX XSAV          ; SAVE X REG.
40B8 B5DA              LDA DIM-1,X
40BA 20F2E2            JSR GIVAYF
40BD 202EED            JSR PTRFAC
40C0 A646              LDX XSAV          ; RECOVER X REG.
40C2 CA                DEX
40C3 E4D9              CPX INDY
40C5 F008              BEQ LABEL3
40C7 A9AC              LDA #","
40C9 205CDB            JSR OUTDO
40CC 4CB240            JMP LABEL1
40CF           LABEL3:
40CF A929              LDA #')'
40D1 205CDB            JSR OUTDO
40D4 A93D              LDA #'='
40D6           ;
40D6           ; NOW PRINT THE NUMERIC OR CHAR VALUE
40D6           ;
40D6 205CDB            JSR OUTDO
40D9 A5D0              LDA TYPE
40DB C903              CMP #$03
40DD F008              BEQ LBEL10        ; TYPE=3,PRINT STRING
40DF 300C              BMI LBEL11        ; TYPE=2,PRINT INTEGER VALUE
40E1           ;
40E1           ; TYPE=5,PRINT REAL VALUE
40E1           ;
40E1 209941            JSR REALOU
40F4 4CF040            JMP NXTSIM
40E7           LBEL10:
40F7 207741            JSR STROUT
40FA 4CF040            JMP NXTSIM
40ED           LBEL11:
40ED 206741            JSR INTOUT
40F0           ;
40F0           ; NXTSIM SETS THE ARRAY POINTERS TO THE NEXT ARRAY
40F0           ;
40F0           NXTSIM:
40F0 20FBDA            JSR CRDO
40F3 2048F9            JSR PRBLK
40F6 18        NXTS1   CLC
40F7 A5D0              LDA TYPE
40F9 6500              ADC POINTL
40FB 8500              STA POINTL
40FD 9002              BCC CONT2
40FF E601              INC POINTH
4101           CONT2:
4101 204641            JSR END           ; IS IT THE END
4104           ;                           OF THE CURRENT ARRAY ?
4104 205641            JSR WAIT
4107           ;
4107           ; SETS CURRENT ARRAY INDEXES
4107           ;
4107 A6D8              LDX INDX
4109 A4D9              LDY INDY
410B           LABEL4:
410B B5DA              LDA DIM-1,X
410D 18                CLC
410E 6901              ADC #$01
4110 95DA              STA DIM-1,X
4112 CA                DEX
4113 B5DA              LDA DIM-1,X
4115 6900              ADC #$00
4117 95DA              STA DIM-1,X
4119 B9DA00            LDA DIM-1,Y
411C D5DB              CMP DIM,X
411E D08B              BNE LABEL2
4120 88                DEY
4121 B9DA00            LDA DIM-1,Y
4124 D5DA              CMP DIM-1,X
4126 D083              BNE LABEL2
4128 A900              LDA #$00
```

(Continued on next page)

```
412A 95DA          STA DIM-1,X
412C 95DB          STA DIM,X
412E CA            DEX
412F 88            DEY
4130 4C0B41        JMP LABEL4
4133               ;
4133               ; PRINT THE NAME OF THE CURRENT ARRAY
4133               ;
4133               PRINTN:
4133 A000                  LDY #$00
4135 B100                  LDA (POINTL),Y
4137 205CDB                JSR OUTDO
413A C8                    INY
413B B100                  LDA (POINTL),Y
413D 297F                  AND #$7F
413F D002                  BNE CONT3
4141 A9A0                  LDA #" "
4143 4C5CDB        CONT3   JMP OUTDO
4146               ;
4146               ; CHECK IF ALL ELEMENTS OF CURRENT ARRAY ARE PRINTED
4146               ;
4146               END:
4146 A500                  LDA POINTL
4148 C5D5                  CMP ENDVAR
414A D009                  BNE RTS1
414C A501                  LDA POINTH
414E C5D6                  CMP ENDVAR+1
4150 D003                  BNE RTS1
4152 4C1D40                JMP LOOP
4155 60            RTS1    RTS
4156               ;
4156               ; ROUTINE FOR START/STOP LISTING
4156               ;
4156               WAIT:
4156 AD00C0                LDA KBOARD
4159 10FA                  BPL RTS1
415B AD10C0                LDA STROBE
415E AD00C0        WAIT1   LDA KBOARD
4161 10FB                  BPL WAIT1
4163 AD10C0                LDA STROBE
4166 60                    RTS
4167               ;
4167               ; ROUTINE FOR PRINTING INTEGER VALUES
4167               ;
4167               INTOUT:
4167 A000                  LDY #$00
4169 B100                  LDA (POINTL),Y      ; GET LOW BYTE
416B AA                    TAX
416C C8                    INY
416D B100                  LDA (POINTL),Y      ; GET HIGH BYTE
416F A8                    TAY                 ; PUT HIGH BYTE IN Y REG.
4170 8A                    TXA                 ; PUT LOW BYTE IN ACCUMULATOR
4171 20F2E2                JSR GIVAYF          ; CONVERT TO FLOATING POINT
4174 4C2EED                JMP PTRFAC          ; PRINT IT
4177               ;
4177               ; ROUTINE FOR PRINTING STRING
4177               ; POINTED BY SPL,SPH OF LENGTH "LEN"
4177               ;
4177               STROUT:
4177 A000                  LDY #$00
4179 B100                  LDA (POINTL),Y
417B F01B                  BEQ RTS2
417D 8504                  STA LEN
417F C8                    INY
4180 B100                  LDA (POINTL),Y
4182 8502                  STA SPL
4184 C8                    INY
4185 B100                  LDA (POINTL),Y
4187 8503                  STA SPH
4189 A000                  LDY #$00
418B               LOOP1:
418B C404                  CPY LEN
418D F009                  BEQ RTS2
418F B102                  LDA (SPL),Y
4191 205CDB                JSR OUTDO
4194 C8                    INY
4195 4C8B41                JMP LOOP1
4198 60            RTS2    RTS
4199               ;
4199               ; ROUTINE FOR PRINTING REAL VALUE
4199               ;
4199               REALOU:
4199 A401                  LDY POINTH
419B A500                  LDA POINTL
419D 20F9EA                JSR MOVEFM
41A0 4C2EED                JMP PTRFAC
```

MICRO

Microbes *(Continued)*

Here is a note from Chuck Wardin, Colorado Springs, Colorado:

Thank you for the fine article and program "Apple Pascal Textfile Lister, (44:100). I bind my listings and this format helps me find the listing I want quickly.

I did come across one problem with the program as printed. It will work for the first textfile only and force one to start the program over to get a second file to list. Below is a simple solution.

PROGRAM READ:

```
End
end
Until Filename = ''
End. (* MAIN PROGRAM *)
```

SHOULD READ:

```
End
end;
close (textfile)
Until Filename = ''
End. (* MAIN PROGRAM *)
```

MICRO

Integer Cross-Reference Utilities

by Lee Reynolds

A cross-reference table is an invaluable aid to development or debugging of a BASIC program. This article and the accompanying program confront the task of generating a complete cross-reference table for Apple Integer BASIC programs.

These Utilities require:

Apple II
with Integer BASIC

Most Apple programmers probahly realize that a computer program is a constantly changing and growing entity. You finish writing a program that does just what you want, and later you think of something else you would like it to do. Perhaps you think of a different technique that will hetter achieve your goal, or — horrors! — you find a bug in your masterpiece. Whatever the reason, most programs need to be modified in some way. Too often the changes must be made long after all the programming details have fled from memory.

This article presents two different cross-reference applications: one prints out a cross-reference of all the line numhers used in a program, and the other does the same for all variable and array names. It is usually much easier to modify a program when you know where every GOTO or GOSUB is going, or in what lines each of your variables and arrays is used.

Applesoft programmers have available a variety of cross-reference utilities that perform these valuahle

Line Number Cross Referencer

```
  5 LOMEM:2500
 10 DIM A$(255):SHLAM=300
 20 CALL -936: PRINT "THIS PROGRAM GENERATES A BINARY FILE": PRINT "WHICH IS
    THE MACHINE LANGUAGE"
 30 PRINT "LINE NUMBER CROSS REFERENCE ROUTINE"
100 A$="0800:A5 CA 85 03 A5 CB 85 04 A9 00 85 0C 20 58 FC A2 00 BD 1E 08 20 E
    D FD E8 E0 12 D0 F5 F0 ": GOSUB SHLAM
110 A$="081D:12 CC C9 CE C5 A3 A0 A0 A0 D2 C5 C6 C5 D2 C5 CE C3 C5 D3 A0 00 B
    1 03 85 09 84 00 C8 B1 ": GOSUB SHLAM
120 A$="083A:03 85 01 C8 B1 03 85 02 A5 CA 85 07 A5 CB 85 08 A0 00 B1 07 85 0
    A C8 B1 07 85 05 C8 B1 ": GOSUB SHLAM
130 A$="0857:07 85 06 A0 03 B1 07 C9 5F F0 4B C9 5C F0 47 C9 24 F0 43 C9 08 F
    0 3F C9 09 F0 41 C9 74 ": GOSUB SHLAM
140 A$="0874:F0 43 C8 C4 0A 90 E1 A5 07 18 65 0A 85 07 A5 08 69 00 85 08 A5 0
    7 C5 4C A5 08 E5 4D 90 ": GOSUB SHLAM
150 A$="0891:B8 A5 03 18 65 09 85 03 A5 04 69 00 85 04 A5 03 C5 4C A5 04 E5 4
    D 90 87 20 BE FD 60 A9 ": GOSUB SHLAM
160 A$="08AE:00 85 0B F0 0A A9 0A 85 0B D0 04 A9 75 85 0B C8 B1 07 C9 B0 90 1
    D C9 BA B0 19 C8 B1 07 ": GOSUB SHLAM
170 A$="08CB:C8 C5 01 D0 11 B1 07 C5 02 D0 0B C8 B1 07 C9 04 90 1D C5 0B F0 1
    9 A5 0B F0 91 C8 B1 07 ": GOSUB SHLAM
180 A$="08E8:C5 0B F0 06 C9 04 90 86 B0 F3 A9 00 85 0B F0 C5 F0 AF A5 00 D0 1
    4 E6 00 20 8E FD A9 00 ": GOSUB SHLAM
190 A$="0905:85 0C A5 01 85 F2 A5 02 85 F3 20 1F E5 A5 0C 18 69 06 C9 24 D0 0
    5 20 8E FD A9 06 85 0C ": GOSUB SHLAM
200 A$="0922:85 24 A5 05 85 F2 A5 06 85 F3 20 1F E5 A9 00 20 AB FC 2C 00 C0 1
    0 18 AD 00 C0 2C 10 C0 ": GOSUB SHLAM
210 A$="093F:C9 A0 F0 06 C9 8D F0 B1 D0 08 2C 00 C0 10 FB 8D 10 C0 4C 7B 08 "
    : GOSUB SHLAM
220 PRINT : PRINT "ROUTINE HAS BEEN POKED INTO MEMORY"
222 PRINT "INSTRUCTIONS:": PRINT " 1. CALL 2048": PRINT " 2. PAUSE DISPLAY WI
    TH SPACE BAR, OR"
224 PRINT " 3. ABORT DISPLAY WITH RETURN KEY"
230 NEW
240 END
300 A$( LEN(A$)+1)="N E88AG": FOR I=1 TO LEN(A$): POKE 511+I, ASC(A$(I))
    : NEXT I: POKE 72,0: CALL -144
310 RETURN
```

Symbol Cross Referencer

```
 10 DIM A$(255):SHLAM=300
 20 CALL -936: PRINT "THIS PROGRAM GENERATES A BINARY FILE": PRINT "WHICH IS
    THE MACHINE LANGUAGE"
 30 PRINT "SINGLE VARIABLE CROSS REFERENCE ROUTINE"
100 A$="300:20 58 FC A0 00 84 09 B1 4A F0 0C C9 40 D0 02 A9 A4 20 F0 FD C8 D0
     F0 84 00 20 8E FD A5 CA ": GOSUB SHLAM
110 A$="031E:85 03 A5 CB 85 04 A0 00 B1 03 85 01 38 E5 00 85 05 90 7C A0 03 C
    4 05 B0 76 B1 03 C9 80 ": GOSUB SHLAM
120 A$="033B:B0 16 C9 5D F0 6C C9 28 D0 0B C8 C4 05 B0 63 B1 03 C9 29 D0 F5 C
    8 D0 E0 C9 C1 B0 04 C8 ": GOSUB SHLAM
130 A$="0358:C8 90 F5 84 06 18 98 65 03 85 07 A5 04 69 00 85 08 A0 00 B1 07 D
    1 4A D0 0B C8 C4 00 D0 ": GOSUB SHLAM
140 A$="0375:F5 B1 07 C9 80 90 12 98 18 65 06 A8 B1 03 C9 80 90 AC C8 C4 05 9
    0 F5 B0 1F A5 09 69 06 ": GOSUB SHLAM
150 A$="0392:C9 24 D0 05 20 8E FD A9 06 85 09 85 24 A0 01 B1 03 85 F2 C8 B1 0
    3 85 F3 20 1F E5 A5 01 ": GOSUB SHLAM
160 A$="03AF:18 65 03 85 03 AA A5 04 69 00 85 04 E4 4C E5 4D B0 03 4C 24 03 2
    0 8E FD 60 ": GOSUB SHLAM
170 PRINT : PRINT "MACHINE LANGUAGE ROUTINE HAS BEEN": PRINT "POKED INTO MEMO
    RY"
172 PRINT "INSTRUCTIONS:": PRINT " 1. TYPE 'CLR'": PRINT " 2. DECLARE SYMBOL
    TO XREF": PRINT " 3. CALL 768"
180 NEW
190 END
300 A$( LEN(A$)+1)="N E88AG": FOR I=1 TO LEN(A$): POKE 511+I, ASC(A$(I))
    : NEXT I: POKE 72,0: CALL -144
310 RETURN
```

functions. For example, the Applesoft Tool Kit (which is part of Apple's DOS Tool Kit) has a symbol cross-referencing capability built into it. Roger Wagner's Apple-Doc package contains routines to perform both types of cross-referencing. Both of these utilities are for Applesoft. What about Integer BASIC?

When I bought my Apple II, most of my early programming was in Integer, so one of my first serious tasks was to write such utilities for Integer. This article contains listings of my results. Both were first written in assembly language; source listings from my assembler are included. Later, I wrote Integer programs that POKEd the machine language routines into memory, using S.H. Lam's method. Listings of these programs are also included.

The line cross-reference routine resides in memory locations \$800 to \$953, while the symbol cross-reference routine extends from \$300 to \$3C7. Consequently, it is possible to have your Integer program and both routines in memory with no addressing conflicts. If you choose to BLOAD the line cross-referencer (rather than RUNning the Integer program that POKEs it into memory), you will have to set LOMEM to some address higher than \$953. This will prevent the routine from getting wiped out by any immediate-mode statements you type in that contain a variable or array name. My Integer program automatically does this in line 5 by setting LOMEM to 2500 before POKEing the routine into that part of memory which is usually reserved for the Integer symbol table.

Both routines will display the cross-reference table on the screen; the line number cross-referencer can be stopped at any time by hitting the space bar. When you want to resume the display, merely press another key. If you want to permanently abort the display, press the return key.

You can, of course, get the tables printed out by doing a PR#1 (or whatever other slot you use) before CALLing the machine language routine.

RUNning the Integer programs will set up the machine language cross-referencers. You then LOAD the Integer program that you want to cross-reference. If you want to perform a line number cross-reference, you start the

Integer BASIC Symbol XREF

```
                     ;1NTEGER BASIC SYMBOL XREF
                     ;
                     ;BY LEE REYNOLDS
                     ;
                     ;ZERO PAGE
                     ;
0000                 SYMLFN   FPZ $00           ;SYMBOL NAME 1ENGTH
0001                 LINLEN   EPZ $01           ;LFNGTH OF TEST L1NE
0003                 LINADR   EPZ $03           ;CURR. PROGR. LINE ADDR.
0005                 LAST     EPZ $05           ;# BYTES TO TEST UP TO IN CURR
. LINE
0006                 CURBYT   EPZ $06           ;Y VALUE FOR CURRENT MATCH
0007                 CURADR   EPZ $07           ;ADDR. OF CURR. MATCH
0009                 HPOS     EPZ $09           ;LAST CURSOR HORIZ. POS.
0800                 ;
0300                          ORG $300
0300                          OBJ $800
0300                 ;
0300 20 58 FC                 JSR $FC58         ;HOME & CLEAR SCREEN
0303 A0 D0                    LDY #$00
0305 84 09                    STY HPOS          ;INIT. CURS. POS.
0307 B1 4A           LOOP1    LDA ($4A),Y       ;SYMBOL PRINT LOOP
0309 F0 0C                    BEQ FNDREF        ;DSP BYTE?
030B C9 40                    CMP #$40          ;@ SIGN?
030D 00 02                    BNE OUTCHR
030F A9 A4                    LDA #$A4          ;CHANGE TO $
0311 20 F0 FD        OUTCHR   JSR $FDF0         ;CHAR. OUTPUT
0314 C8                       INY               ;INCR. # CHARS.
0315 D0 F0                    BNE LOOP1         ;ALWAYS
0317 84 00           FNDREF   STY SYMLFN        ;SAVE SYM. LENGTH
0319 20 8E FD                 JSR $FD8E         ;OUTPUT CARRIAGE RETURN
031C A5 CA                    LDA $CA           ;LOMEM, LOW
031E 85 03                    STA LINADR
0320 A5 CB                    LDA $CB           ;LOMEM, HIGH
0322 85 04                    STA LINADR+1
0324 A0 00           LOOP2    LDY #$00          ;LINE SEARCH LOOP
0326 B1 03                    LDA (LINADR),Y    ;LINE LENGTH
0328 85 01                    STA LINLEN        ;SAVE LINE LENGTH
032A 38                       SEC
032B E5 00                    SBC SYMLFN        ;SUBTRACT SYM. LENGTH
032D 85 05                    STA LAST          ;SAVE PTR TO LAST BYTE TO TEST

032F 90 7C                    BCC NXTLIN
0331 A0 03                    LDY #$03          ;GET PAST LINE #
0333 C4 05           TSTOKN   CPY LAST          ;FIND NON-TOKEN LOOP
0335 B0 76                    BCS NXTLIN        ;Y>=PTR TO LAST?
0337 B1 03                    LDA (L1NADR),Y
0339 C9 80                    CMP #$80
033B B0 16                    BCS TSTNUM        ;>=$80?
033D C9 5D                    CMP #$5D
033F F0 6C                    BEQ NXTLIN        ;REM TOKEN?
0341 C9 28                    CMP #$28          ;BEGIN QUOTE?
0343 D0 0B                    BNE NXTBYT
0345 C8              LOOP3    INY               ;FIND QUOTE LOOP
0346 C4 05                    CPY LAST
0348 B0 63                    BCS NXTLIN        ;DONF WITH LINE?
034A B1 03                    LDA (LINADR),Y
034C C9 29                    CMP #$29          ;END QUOTE?
034E D0 F5                    BNE LOOP3
0350 C8              NXTBYT   INY
0351 D0 E0                    BNE TSTOKN        ;ALWAYS
0353 C9 C1           TSTNUM   CMP #$C1
0355 B0 04                    BCS ALPHA
0357 C8                       INY               ;SKIP OVER 2ND BYTE
0358 C8                       INY               ;SK1P OVER 1ST BYTE OF INTEGER

0359 90 F5                    BCC NXTBYT        ;ALWAYS
035B 84 06           ALPHA    STY CURBYT        ;SAVE PTR TO CURR. BYTE
035D 18                       CLC
035E 98                       TYA
035F 65 03                    ADC LINADR
0361 85 07                    STA CURADR
0363 A5 04                    LDA LINADR+1
0365 69 D0                    ADC #$00
0367 85 08                    STA CURADR+1
0369 A0 00                    LDY #$00
036B B1 07           LOOP4    LDA (CURADR),Y    ;TEST SYMBOL MATCH LOOP
036D D1 4A                    CMP ($4A),Y
036F 00 0B                    BNE FNDTOK        ;NO MATCH. 0O FIND TOKEN
0371 C8                       INY
0372 C4 00                    CPY SYMLEN        ;COMPARE TO SYMBOL LENGTH
0374 D0 F5                    BNE LOOP4
```

(Continued)

Integer BASIC Symbol XREF *(continued)*

```
0376 B1 07              LDA (CURADR),Y     ;NEXT BYTE TOKEN?
0378 C9 80              CMP #$80
037A 90 12              BCC FOUND
037C 98       FNDTOK    TYA
037D 18                 CLC
037E 65 06              ADC CURBYT
0380 A8                 TAY
0381 B1 03    LOOP5     LDA (LINADR),Y     ;FIND TOKEN LOOP
0383 C9 80              CMP #$80
0385 90 AC              BCC TSTOKN         ;<$80?
0387 C8                 INY
0388 C4 05              CPY LAST
038A 90 F5              BCC LOOP5          ;>=LAST VALUE TO TEST?
038C B0 1F              BCS NXTLIN         ;ALWAYS
038E A5 09    FOUND     LDA HPOS
0390 69 06              ADC #$06
0392 C9 24              CMP #$24           ;REACHED 36?
0394 D0 05              BNE PRT
0396 20 8E FD           JSR $FD8E          ;CARRIAGE RETURN
0399 A9 06              LDA #$06
039B 85 09    PRT       STA HPOS           ;SAVE CURSOR POSITION
039D 85 24              STA $24            ;MOVE CURSOR
039F A9 01              LDA #$01
03A1 B1 03              LDA (LINADR),Y     ;REFERENCING LINE #, LO
03A3 85 F2              STA $F2            ;PASS TO SUBR.
03A5 C8                 INY
03A6 B1 03              LDA (LINADR),Y     ;HIGH BYTE
03A8 85 F3              STA $F3            ;PASS TO SUBR.
03AA 20 1F E5           JSR $E51F          ;PRINT LINE #
03AD A5 01    NXTLIN    LDA LINLEN         ;LINE LENGTH
03AF 18                 CLC
03B0 65 03              ADC LINADR
03B2 85 03              STA LINADR
03B4 AA                 TAX
03B5 A5 04              LDA LINADR+1
03B7 69 00              ADC #$00
03B9 85 04              STA LINADR+1       ;NEXT LINE# ADDR.
03BB E4 4C              CPX $4C            ;REACHED HIMEM?
03BD E5 4D              SBC $4D
03BF B0 03              BCS EXIT
03C1 4C 24 03           JMP LOOP2
03C4 20 8E FD EXIT      JSR $FD8E
03C7 60                 RTS
03C8                    END
```

Integer BASIC Line XREF

```
0800                    ;INTEGER BASIC LINE XREF
0800                    ;
0800                    ;BY LEE REYNOLDS
0800                    ;
0000          FLAG1     EPZ $00            ;FLAG:=1 WHEN 1ST REF.
0001          CURLIN    EPZ $01            ;CURRENT LINE #, WHOSE REF'S A
RE BEING SEARCHED
0003          CURADR    EPZ $03            ;ADDRESS OF CURLIN
0005          SRCHLN    EPZ $05            ;CURRENT LINE BEING SEARCHED F
OR REF'S
0007          SRCHAD    EPZ $07            ;ADDRESS OF SRCHLN
0009          LENREF    EPZ $09            ;LENGTH OF REFERENCED LINE
000A          LENSEA    EPZ $0A            ;LENGTH OF LINE BEING SEARCHED
000B          FLAG2     EPZ $0B            ;FLAG: 0=GOTO,$A=DEL,$75=LIST
000C          LSTPOS    EPZ $0C            ;LAST HORIZ. CURSOR POS.
0800                    ;
0300                    ORG $300
0800                    ORG $800
0800                    ;
0800 A5 CA    BEGIN     LDA $CA            ;PROG. START, LOW
0802 85 03              STA CURADR
0804 A5 CB              LDA $CB            ;PROG. START, HIGH
0806 85 04              STA CURADR+1
0808 A9 00              LDA #$00
080A 85 0C              STA $0C            ;INIT. LAST CH
080C 20 58 FC           JSR $FC58          ;HOME & CLEAR SCREEN
080F A2 00              LDX #$00
0811 BD 1E 08 LOOP      LDA TITLE,X
```

(Continued)

display by means of "CALL 2048." If you want to perform the symbol cross-reference, it's a bit more complicated:

1. Type CLR to clear the symbol table. This is necessary because I chose to have my routine perform its cross-reference on only one symbol at a time, and it is always the first one declared.

2. Declare the symbol you want to cross-reference. Thus, if you wanted to find all references of a variable named PLAYER, you would type in a statement such as PLAYER=0. If you wanted to search for an array named BOARD, then a statement like DIM BOARD(64) would do. When cross-referencing a string array, you must also declare the symbol by means of a DIM statement.

3. Activate the display by means of "CALL 768." When you want to cross-reference another variable or array name, begin again from step 1.

Remember that if a non-array variable has the same name as an array, its value is stated in element 0 of the array. Thus, if you have a variable called GAME and also an array called GAME, the value of the variable is saved in GAME(0). This interesting quirk of that language means that my symbol cross-referencer will cross-reference *both* usages at once.

If you understand assembly language, you may find it interesting to delve into the source listings; the comments are fairly complete, so it shouldn't be difficult to understand, *if* you are aware of how Integer BASIC stores program lines in memory. (See the Nov./Dec. 1979 issue of *Call —A.P.P.L.E.*)

Both Integer programs NEW themselves out of memory after running, and — as mentioned before — the line number cross-referencer program must begin by setting LOMEM. Consequently, line 180 in the symbol cross-referencer is "illegal." The same goes for lines 5 and 230 in the line cross-referencer. In order to type these lines in, you will have to go to a bit of trouble. One method is to use Ray McVay's Integer BASIC Post-Editor program (see the March/April 1980 issue of *Call —A.P.P.L.E.*). If you don't have this program available, the changes can be implemented using the following procedures.

Integer Symbol Xref Program Procedure

1. Type in this statement before any others:
   180 PRINT

2. Go into the monitor by means of CALL −151

3. Type CA.CB

You will see something like this:

*00CA− FB 95

This is telling you that locations $CA and $CB contain the values $FB and $95. If you combine the two values into one 4-digit hex number, after switching their order, you will get the memory location $95FB where line number 180 begins. The entire memory representation of this line will be the following sequence of hex values (which, in this case, you can display by means of the monitor command 95FB.95FF):

05 B4 00 63 01

That "63" is what BASIC stores in place of the word "PRINT", which you typed in on line 180. Change that value to the token for the key word "NEW." In this example, 95FE:0B accomplishes this.

When you have succeeded in getting BASIC to accept an illegal statement containing the word "NEW", you must go back to BASIC by means of Control-C (return), and type in the rest of the Integer program.

Integer Line Xref Program Procedure

You will have to go through a very similar process to get lines 5 and 230 into the program.

1. Type in these lines first:
   5 PRINT 2500
   230 PRINT

2. Go into the monitor, by CALL −151

3. Type CA.CB. My 48K system displays:
   *00CA− F3 95

So line number 5 starts at location $95F3. You can display both lines by means of:

95F3.95FF

You will see these hex values:

08 05 00 62 B2 C4 09 01 05 E6 00 63 01

Integer BASIC Line XREF *(continued)*

```
0814 20 ED FD                JSR $FDED          ;OUTPUT CHAR.
0817 E8                      INX
0818 E0 12                   CPX #$12           ;END OF TITLE?
081A D0 F5                   BNE LOOP
081C F0 12                   BEQ INIT1
081E 4C 49 4E       TITLE    ASC 'LINE#   REFERENCES'
0821 45 23 20
0824 20 20 52
0827 45 46 45
082A 52 45 4E
082D 43 45 53
0830 A0 00          INIT1    LDY #$00
0832 B1 03                   LDA (CURADR),Y     ;LENGTH OF LINE
0834 85 09                   STA LENREF
0836 84 00                   STY FLAG1          ;CLEAR FLAG: NO REF'S
0838 C8                      INY
0839 B1 03                   LDA (CURADR),Y     ;LINE TO FIND, LOW
083B 85 01                   STA CURLIN
083D C8                      INY
083E B1 03                   LDA (CURADR),Y     ;LINE TO FIND, HIGH
0840 85 02                   STA CURLIN+1
0842 A5 CA                   LDA $CA            ;PROG. START, LOW
0844 85 07                   STA SRCHAD         ;SEARCH START, LOW
0846 A5 CB                   LDA $CB            ;PROG. START, HIGH
0848 85 08                   STA SRCHAD+1       ;SEARCH START, HIGH
084A A0 00          INIT2    LDY #$00           ;LINE LOOP
084C B1 07                   LDA (SRCHAD),Y     ;LENGTH OF LINE
084E 85 0A                   STA LENSEA
0850 C8                      INY
0851 B1 07                   LDA (SRCHAD),Y     ;LINE# SEARCHING, LO
0853 85 05                   STA SRCHLN
0855 C8                      INY
0856 B1 07                   LDA (SRCHAD),Y     ; "        "    , HIGH
0858 85 06                   STA SRCHLN+1
085A A0 03                   LDY #$03           ;GET PAST LINE #
085C B1 07          SEARCH   LDA (SRCHAD),Y     ;GET CURR. BYTE
085E C9 5F                   CMP #$5F           ;GOTO TOKEN?
0860 F0 4B                   BEQ GOTO
0862 C9 5C                   CMP #$5C           ;GOSUB TOKEN?
0864 F0 47                   BEQ GOTO
0866 C9 24                   CMP #$24           ;THEN TOKEN?
0868 F0 43                   BEQ GOTO
086A C9 08                   CMP #$08           ;RUN TOKEN?
086C F0 3F                   BEQ GOTO
086E C9 09                   CMP #$09           ;DELETE TOKEN?
0870 F0 41                   BEQ DEL
0872 C9 74                   CMP #$74           ;LIST TOKEN?
0874 F0 43                   BEQ LIST
0876 C8             NXTBYT   INY
0877 C4 0A                   CPY LENSEA         ;DONE WITH LINE?
0879 90 E1                   BCC SEARCH
087B A5 07          NXTLI    LDA SRCHAD         ;ADDR. OF LINE SEARCHING
087D 18                      CLC
087E 65 0A                   ADC LENSEA         ;LENGTH
0880 85 07                   STA SRCHAD         ;NEXT LINE ADDR.
0882 A5 08                   LDA SRCHAD+1
0884 69 00                   ADC #$00
0886 85 08                   STA SRCHAD+1
0888 A5 07                   LDA SRCHAD
088A C5 4C                   CMP $4C            ;COMPARE TO HIMEM
088C A5 08                   LDA SRCHAD+1
088E E5 4D                   SBC $4D
0890 90 B8                   BCC INIT2
0892 A5 03                   LDA CURADR         ;ADDR. OF TEST LINE
0894 18                      CLC
0895 65 09                   ADC LENREF         ;LENGTH
0897 85 03                   STA CURADR         ;NEXT TEST LINE ADDR.
0899 A5 04                   LDA CURADR+1
089B 69 00                   ADC #$00
089D 85 04                   STA CURADR+1
089F A5 03                   LDA CURADR
08A1 C5 4C                   CMP $4C            ;END OF PROGRAM?
08A3 A5 04                   LDA CURADR+1
08A5 E5 4D                   SBC $4D
08A7 90 87                   BCC INIT1
08A9 20 8E FD       EXIT     JSR $FD8E          ;PRINT CARR. RET.
08AC 60                      RTS                ;GO BACK TO BASIC
08AD A9 00          GOTO     LDA #$00
08AF 85 0B                   STA FLAG2          ;FLAG TESTING GOTO
08B1 F0 0A                   BEQ TSTLIN
08B3 A9 0A          DEL      LDA #$0A
08B5 85 0B                   STA FLAG2          ;FLAG TESTING DELETE
08B7 D0 04                   BNE TSTLIN
```

(Continued)

Integer BASIC Line XREF *(continued)*

```
08B9 A9 75      105  LIST     LDA #$75
08BB 85 0B      106           STA FLAG2          ;FLAG TESTING LIST
08BD C8         107  TSTLIN   INY                ;BYTE AFTER GOTO, ETC.
08BE B1 07      108           LDA (SRCHAD),Y
08C0 C9 B0      109           CMP #$B0
08C2 90 1D      110           BCC TESTB          ;<$B0?
08C4 C9 BA      111           CMP #$BA
08C6 B0 19      112           BCS TESTB          ;>$B9?
08C8            113  ;
08C8            114  ;BYTE VALUE BETWEEN $B0 & $B9 SAYS
08C8            115  ;NEXT TWO BYTES ARE INTEGER #
08C8            116  ;
08C8 C8         117           INY
08C9 B1 07      118           LDA (SRCHAD),Y
08CB C8         119           INY
08CC C5 01      120           CMP CURLIN         ;LOW BYTE OF TEST LINE?
08CE D0 11      121           BNE TESTB
08D0 B1 07      122           LDA (SRCHAD),Y
08D2 C5 02      123           CMP CURLIN+1       ;HIGH BYTE?
08D4 D0 0B      124           BNE TESTB
08D6 C8         125           INY                ;POINT TO BYTE AFTER INTEGER #

08D7 B1 07      126           LDA (SRCHAD),Y
08D9 C9 04      127           CMP #$D4
08DB 90 1D      128           BCC PRINT          ;PRINT IT'S SEMICOLON OR END-O
F-LINE TOKEN
08DD C5 0B      129           CMP FLAG2          ;TYPE OF COMMA TOKEN
08DF F0 19      130           BEQ PRINT
08E1 A5 0B      131  TESTB    LDA FLAG2
08E3 F0 91      132           BEQ NXTBYT         ;GOTO, ETC.

08E5 C8         133  FNDCOM   INY                ;FIND COMMA OR STATEMENT SEPAR
ATOR
08E6 B1 07      134           LDA (SRCHAD),Y
08E8 C5 0B      135           CMP FLAG2
08EA F0 06      136           BEQ FOUND
08EC C9 04      137           CMP #$04
08EE 90 86      138           BCC NXTBYT
08F0 B0 F3      139           BCS FNDCOM
08F2 A9 D0      140  FOUND    LDA #$D0           ;FLAG FOR 1ST REF.
08F4 85 0B      141           STA FLAG2
08F6 F0 C5      142           BEQ TSTLIN         ;ALWAYS
08F8 F0 AF      143  OUT      BEQ EXIT           ;ALWAYS
08FA A5 D0      144  PRINT    LDA FLAG1          ;FLAG FOR 1ST REF.
08FC D0 14      145           BNE PRTREF         ;NOT FIRST REF?
08FE E6 00      146           INC FLAG1          ;FLAG 1ST REF. FOUND
0900 20 8E FD   147           JSR $FD8E          ;PRINT CARR. RET.
0903 A9 D0      148           LDA #$00           ;BEGIN NEW LINE
0905 85 0C      149           STA LSTPOS
0907 A5 01      150           LDA CURLIN         ;TEST LINE#, LOW
0909 85 F2      151           STA $F2            ;PASS TO ROUTINE
090B A5 02      152           LDA CURLIN+1       ;TEST LINE#, HIGH
090D 85 F3      153           STA $F3            ;PASS
090F 20 1F E5   154           JSR $E51F          ;PRINT TEST LINE#
0912 A5 0C      155  PRTREF   LDA LSTPOS         ;LAST CURSOR HORIZ.
0914 18         156           CLC
0915 69 06      157           ADC #$06
0917 C9 24      158           CMP #$24           ;REACHED POS. 36?
0919 D0 05      159           BNE PRT
091B 20 8E FD   160           JSR $FD8E          ;CARR. RET.
091E A9 06      161           LDA #$06
0920 85 0C      162  PRT      STA LSTPOS
0922 85 24      163           STA $24            ;MOVE CURSOR
0924 A5 05      164           LDA SRCHLN         ;REFER. LINE #, LOW
0926 85 F2      165           STA $F2
0928 A5 06      166           LDA SRCHLN+1       ;HIGH
092A 85 F3      167           STA $F3
092C 20 1F E5   168           JSR $E51F          ;PRINT REF. LINE #
092F A9 00      169           LDA #$D0
0931 20 A8 FC   170           JSR $FCA8          ;MAKE A LONG DELAY
0934 2C 00 C0   171           BIT $C000          ;TEST KBD. STROBE
0937 10 18      172           BPL ENDL1          ;NOTHING TYPED?
0939 AD D0 C0   173           LDA $C000          ;GET KEY TYPED
093C 2C 10 C0   174           BIT $C010          ;CLR KBD. STROBE
093F C9 A0      175           CMP #$A0           ;IS IT A SPACE?
0941 F0 06      176           BEQ STOP           ;GO STOP PRINTING
0943 C9 8D      177           CMP #$8D           ;CARR. RET.?
0945 F0 B1      178           BEQ OUT            ;GO END PROGRAM
0947 00 08      179           BNE ENDL1
0949 2C 00 C0   180  STOP     BIT $C000          ;TEST STROBE
094C 10 FB      181           BPL STOP           ;WAIT FOR KEYIN
094E 8D 10 C0   182           STA $C010          ;CLR STROBE
0951 4C 7B 08   183  ENDL1    JMP NXTL1
0954            184           END
```

You must change the "62" to an "11", and the "63" to a "0B". On my system, these monitor commands would do that:

```
95F6:11
95FE:0B
```

4. Now go back to BASIC and enter the rest of the program.

If you don't know anything about the hexadecimal numbering system, or about the monitor commands, you should leave out line 230 of my Integer line cross-referencer, and NEW the program out of memory yourself in immediate mode, after RUNning it. Also, leave out lines 5 and 180 in the other program, set LOMEM to 2500 before you RUN it, then NEW it out afterwards.

Lee Reynolds, a computer programmer for 15 years, owns an Apple II. He has published almost two dozen articles in magazines such as *MICRO, Call -A.P.P.L.E.*, and *Softalk*. Reynolds may be contacted at 5760 N.W. 60 Ave., Apt. B-101, Ft.Lauderdale, FL 33319.

MICRO

MICRO

6502 Bibliography

Dr. William R. Dial, 438 Roslyn Ave., Akron, OH 44320

1224. Commodore Interface (May, 1981)

Anon., "Micro-Mainframe," pg. 5.
The new CBM micro-mainframe computer contains both 6502 and 6809 processors, 96K RAM and 36K ROM, etc.

Berk, Emily, "An Introduction to BASIC, Machine Code, and Assembly Language," pg. 15-17.
Part two of an instructional series for CBM owners.

1225. Spreadsheet 1, No. 4 (May, 1981)

Dawson, Peter, "Visitip No. 9," pg. 1.
How to save only a portion of a matrix on VisiCalc.

Bostater, John, "Greater Than/Equals or Less Than/Equals," pg. 2.
A search technique for VisiCalc.

Shen, Robert, "Visitip No. 10," pg. 3.
Some suggestions on using VisiCalc.

Thomas, Elizabeth, "An Algorithm That Works Like 'If-Then'," pg. 6.
A technique for VisiCalc and a budget matrix example.

1226. The G.R.A.P.E. Vine 2, No. 4 (May, 1981)

Lawson, Steve, "Disk/Apple Configuration," pg. 7.
A program to identify slave/master status, Apple memory size, ROM or RAM Applesoft, name of Hello program on last booted disk.

1227. L.A.U.G.H.S. 3, No. 5 (May, 1981)

Roe, David, "The Language Card and the 'Old' Monitor," pg. 4.
How to enter the old monitor with the language card installed on your Apple.

1228. Purser's Magazine No. 12 (Winter, 1981)

Staff, "Program Reviews," pg. 24-94.
Some 70 pieces of software for the Apple II and the TRS-80 are reviewed.

1229. Peelings II 2, No. 3 (May/June, 1981)

Staff, "Apple Software Evaluation," pg. 8-41.
Twenty-two pieces of Apple software are reviewed and evaluated. Included are DB Master, CCA Data MGMT System, VisiCalc, VisiCaids, Introstat, utilities including Apple-Doc, Curve Fitter, Super Shape Draw, Higher Graphcis II, languages such as Dynasoft Pascal, Tiny Pascal, FORTH 1.7, Super FORTH, App-L-Isp, APPilot II, and games including *Ultracheckers*, *Golden Mountain*, *Apple-Oids*, *Reversal*, *Hi-Res Cribbage*, and *Astroscope*.

1230. AppleGram 3, No. 5 (May, 1981)

Zant, Robert F., "Apple Writer," pg. 4-6.
Some notes on this simple word processor for the Apple.

Sander-Cederlof, Bob, "Measurement Conversion Program," pg. 19-20.
How many teaspoons in a liter and the like.

Anon, "Epson MX-80 Table," pg. 22.
An aid to Epson users.

(Continued on page 100)

WHO ARE YOU THIS TIME?

Are you the seeker of lost treasures in an enchanted realm of magical beings? Perhaps you're an astronaut, thousands of light years from earth, searching the galaxy's rim for the fabulous treasures and advanced technologies of a long-dead civilization. Maybe you're the plunderer of ancient pyramids in a maddeningly dangerous land of crumbling ruins, and trackless desert wastes.

Imagine.

You wish to soar to other worlds, to behold wonders never beheld by mortal eyes, to dream unrestrained to the furthest limits of your imagination.

Imagine now that you can.

The ADVENTURE SERIES by Scott Adams has been reviewed by every major microcomputing magazine, receiving only the highest praise for its mind-puzzling challenge & refreshing originality. Tens of thousands of adults and children have matched wits with the Adventure Master. Now, it's your turn. If you crave the challenge and panorama of the exotic, touched with sparkling humor, then the Adventure Series is for you.

With the incredible Adventure Series by Scott Adams:
YOU ARE WHOMEVER YOU WANT TO BE WHENEVER YOU WANT TO BE!

	SINGLE TAPE	12 PACK TAPE	SINGLE DISK	TRIPLE DISK	12 PACK DISK
APPLE	$19.95 24K	$129.95† 24K	$29.95* 48K	$69.95* 48K	—
ATARI	$19.95 24K	$129.95† 24K	—	$39.95 40K	$129.95† 48K
CP/M (Z-80)	—	—	—	—	$129.95†
TRS-80 MOD I&3	$19.95 16K	$129.95† 16K	—	$39.95 32K	$129.95† 32K
TRS-80 MOD 2	—	—	—	—	$129.95

PRICES SUBJECT TO CHANGE WITHOUT NOTICE

Adventure Hint Book (All 12)	$ 7.95
Adventure Hint Sheet (Single)	1.00
Commodore VIC (Cartridge)	39.95
T.I. 99/4 (Tape or Disk)	29.95

Requires Adventure Command Module

*Scott Adams' Graphic Adventure Series Apple Disks-Full Color Hi Res Graphics
†12 Adventure Hint Book Incl.

SHIPPING & HANDLING ARE EXTRA

Adventure INTERNATIONAL
A DIVISION OF SCOTT ADAMS, INC.
© 1982

BOX 3435, LONGWOOD FL 32750
(305) 862-6917 (QUESTIONS)
ORDER FROM YOUR FAVORITE DEALER or CALL TOLL FREE (800) 327-7172
(ORDERS ONLY PLEASE) #119

6502 Bibliography *(Continued)*

1231. Apple/Sass 3, No. 4 (May, 1981)

Anon., "Give Me a BRK!" pg. 5-6.
What happens on an Apple when the BRK command is encountered.

Fan, Kenny C., "Super Frame," pg. 12-17.
Two machine language programs that can add special effects to your titles or programs, making a frame that moves around the sides of the screen.

1232. The C.I.D.E.R. Press 3, No. 3 (May/June, 1981)

Hall, John, "Bit Decoding Routines," pg. 2.
Routines in Integer and Applesoft BASIC that decode a byte value and decode it into the eight individual bits that form the binary number.

1233. OSIO Newsletter 3, No. 6 (June, 1981)

Goodhart, Brian, "Unravl: A Strategy," pg. 1-2.
Use of the OSI Flag 7 in figuring out how someone else's BASIC program works.

Compton, Radford, "Money Mode Fix," pg. 6.
A fix for a bug in the formatting of small calculated sums of dollars and cents, in OS-65U.

1234. The Harvest 2, No. 10 (June, 1981)

Schaffer, Jay, "The Apple Throttle," pg. 7.
Control the speed of your Apple BASIC listings with the game paddle.

1235. Saturn Softnews 1, No. 1 (June, 1981)

Staff, "Extended SYM-BASIC," pg. 2.
A review and listing of all new commands for the SYM.

1236. Apple-Dayton Newsletter 2, No. 6 (June, 1981)

Matthews, John, "On Cycles and Such," pg. 10-12.
A biorhythm program for the Apple.

1237. Atari Computer Enthusiasts 2, Issue 6 (June, 1981)

Anon., "Using USR," pg. 4-5.
An introduction to the use of machine language on the Atari, with examples of using the USR function.

Oekers, Stan, "Lunar V," pg. 7-8.
A game for the Atari.

Ekberg, Michael, "Load and Save Binary Files," pg. 9.
An Atari routine to load or save a binary file from BASIC.

1238. Nibble 2, No. 3 (June. 1981)

Hatcher, Rich, "The Mad, Mad, Mad, Mad Cube," pg. 19-25.
A Rubik's Cube-type program for the Apple.

Koeritz, Chris, "The Apple Hi-Res Clock," pg. 27-28.
A clock face program using Apple graphics.

Black, Preston R., "Disk Snooping — Part II," pg. 25-39.
Using the tools on the DOS, Disk Directory, File Sectors, Track/Sector List, VTOC, etc.

Mossberg, Sandy, "Applesoft Linefinder," pg. 63-67.
A debugging aid for Applesoft programmers.

Mottola, R.M., "Using the USR Function for Address Referencing," pg. 83-87.
The USR function and how it works on the Apple.

Allen, David P., "The Little Line Eater," pg. 87.
How to put disappearing lines into your Applesoft programs.

Reynolds, William III, "Converting 'Muffin' into 'Demuffin'," pg. 87.
A simple job with the instructions given here.

![MICRO]

6809 Bibliography

Dr. William R. Dial, 438 Roslyn Ave., Akron, Ohio 44320

Time Marches On

Dr. William Dial's 6502 Bibliography has played a major role in making bibliographical information available to 6502 users. But now that the 6502 is a mature processor, we at MICRO believe that most 6502 users need selectivity more than comprehensiveness. Therefore, the 6502 Bibliography in MICRO will in future selectively list a much smaller number of the better 6502 articles.

Users of the 6809 processor, however, do need the kind of comprehensive coverage that MICRO used to give the 6502. Therefore, MICRO will now start a comprehensive 6809 bibliography, to be published in installments as material accumulates. If any readers are aware of 6809 material we are missing, please contact Dr. Dial or the MICRO staff.

We feel that this combination—selective 6502 and comprehensive 6809 coverage—will serve our readers best.

1. **Softalk 1, No. 9 (May, 1981)**

Anon., "The Mill," pg. 25.
A review of a new Apple peripheral board based on the new 6809E microprocessor, offering Apple users a 8/16-bit architecture, direct page register, extensive addressing modes, fast speed, etc.

2. **KB Microcomputing 5, No. 5, Issue 53 (May, 1981)**

Stark, Peter A., "6800's Best-Kept Secrets," pg. 56-66.
Included in the review of various editors and word processors are the TSC Text Editor/Text Processor for the 6809 systems, all-in-one editor/processor for the 6809, and the Stylograph editor/processor for the 6809.

3. **Compute! 3, No. 5, Issue 12 (May, 1981)**

Lock, Robert, "Introducing Super PET," pg. 4-8.
A new CBM micro has 134K mixed RAM and ROM with both 6502 and 6809 processors and separate ROM operating systems and several languages.

4. **MICRO No. 37 (June, 1981)**

Tripp, Robert M., "It's Time to Stop Dreaming, Part 1," pg. 7-9.
A description of the features of the 6809 microprocessor, a possible candidate to update the 6502.

5. **Abacus II 3, Issue 5 (May, 1981)**

Anon., "What's A Hitachi?" pg. 3.
The Hitachi MB 6890 is a new microcomputer based on the 6809 microprocessor running at 1 MHz.

6. **KB Microcomputing 5, No. 7, Issue 55 (July, 1981)**

Rawson, David R., "Clock/Calendar for the 6809," pg. 132-141.
Hardware and software for implementing a clock on 6809 systems.

7. **BYTE 6, No. 7 (July, 1981)**

Scales, Hunter, "Multiprocessing with Motorola's MC6809E," pg. 136-156.
How to use two or more microprocessors sharing common resources, each working on a part of the problem.

Anon., "6809 Cross Assembler," pg. 438.
The XASM 6809 is a commercially available cross-assembler written in FORTRAN IV.

8. **MICRO No. 38 (July, 1981)**

Tripp, Robert M., "It's Time to Stop Dreaming, Part 2," pg. 27-30.
Part 2 describes some of the improvements which are provided by this chip. These include long branches to any location, a branch to subroutine instruction with relative branching, addressing relative to the program counter, and a load effective address instruction.

Wright, Loren W., "PET Vet," pg. 91.
A new assembler for 8K PETs — a new 6809-based micro from Commodore (Micro-Mainframe or "Super PET").

9. **Dr. Dobb's Journal 6, Issue 7, No. 57 (July, 1981)**

Gordon, H.T., "About the Motorola 6809," pg. 6-9.
Discussion of the characteristics of the 6809 microprocessor and its probable impact on personal computers.

10. **Interface Age 6, Issue 8 (August, 1981)**

Baker, Al, "Game Corner," pg. 24-28.
A tutorial on color graphics with the 6809-based TRS-80 Color Computer.

11. **Personal Computing 5, No. 6 (August, 1981)**

Anon., "Some Japanese Personal Computers," pg. 100.
In a table of new Japanese micros it is revealed that the 6809 microprocessor is used in the Hitachi 6890, the Canon BX-3 and the Canon CX-1.

12. **BYTE 6, No. 8 (August, 1981)**

Miatkowski, Stan, "The Japanese Computer Invasion," pg. 200-220.
The Fujitsu Micro-8 uses twin 6809 microprocessors to greatly increase speed.

13. **KB Microcomputing 5, No. 8, Issue 56 (August, 1981)**

Baker, Robert W., "Petpourri," pg. 10-16.
The CBM 8032 color computer and the new CBM Micro-Mainframe (based on the 6809) are described.

14. **MICRO No. 39 (August, 1981)**

Tripp, Robert M., "It's Time to Stop Dreaming, Part 3," pg. 16-18.
Part 3 of this series on the 6809 microprocessor describes the instruction set in detail, comparing it to the familiar 6502 set.

15. **Rubber Apple Newsletter 4**

Anon., "6502 *vs.* 6800 *vs.* 6809," pg. 7-12.
A comparison of three microprocessors.

16. **KB Microcomputing 5, No. 9, Issue 57 (September, 1981)**

Vose, G. Michael, "Exploring the MC6809," pg. 25-30.
A description of the 6809 microprocessor.

17. **MICRO No. 40 (September, 1981)**

Tripp, Robert M., "It's Time to Stop Dreaming, Part 4," pg. 20-22.
A discussion of the addressing modes of the 6809, comparing the 6809 with the 6502, with special emphasis on the greatly expanded options for the 6809.

MICRO New Publications

Mike Rowe
New Publications
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Intimate Instructions in Integer BASIC, by Brian D. Blackwood and George H. Blackwood. Howard W. Sams and Co., Inc. (4300 West 62nd St., Indianapolis, IN 46268), 1981, 158 pages, 5¼ × 8½ inches, paperback.
ISBN: 0-672-21812-7 $7.95

Although written for Apple II users, this book can apply, with modifications, to other microcomputers using BASIC. In a lesson-type format, each chapter provides definitions, the basic fundamentals of programming techniques, and self-testing exercises.

CONTENTS: *Introduction; Clear the VDM Screen; Load and Save Program; Programming and Print Rules; Operators; Truncation and Integers; Simulated Reals; Catch-All; Flowcharting; Loops and Counting Variables; Rule of Default and Decision Statements; General Outline of Programs; Playing Computer; Subscripted Variables; MIN-MAX and SORT; Strings and GOSUB; Functions; Efficient Programming; Graphics; Menu and Flag; Games; Appendix; Index.*

Pascal: A Problem Solving Approach, by Elliot B. Koffman. Addison-Wesley (Reading, MA 01867), 1982, 6 × 9 inches, paperback.
ISBN: 0-201-10341-9 $14.95

This book emphasizes the structured, step-by-step design of computer programs. Both beginning programmers and those experienced in other languages will learn programming techniques, problem-solving skills, and UCSD Pascal.

CONTENTS: *Introduction to Computers and Programming*—Introduction; Computer organization; Programs and programming languages; Introduction to Pascal; Using the computer; Additional input and output features; Introduction to data types; Summary; Programming problems. *Problem Solving with the Computer*—Introduction; Problem analysis; Description of the problem solution; Algorithms involving decisions; Algorithms with loops; Implementing the algorithm; Summary; Programming problems. *Fundamental Control Statements*—Introduction to control statements; The IF statement; The WHILE statement; Application of control statements; The FOR statement; The widget inventory control problem; Debugging and testing programs; Common programming errors; Summary; Programming problems. *Standard, Scalar, and Subrange Data Types*—Introduction; Numeric data types—REAL and INTEGER; Functions in arithmetic expressions; Boolean variables, expressions and operations; String variables; Character variables and functions; More on input; Scalar and subtange data types; Numerical errors; Common programming errors; Summary; Programming problems. *Intermediate Control Structures*—Introduction; Multiple-alternative decisions; Top-down programming and functions; Procedures; Application of top-down design; Scope of an identifier; Common programming errors; Summary; Programming problems. *Arrays and Strings*—Introduction; Declaring arrays; Array subscripts; Manipulating array elements; Manipulating entire arrays; Partially filled arrays; Arrays of strings; Manipulating character strings; Common programming errors; Summary; Programming problems. *Records and Sets*—Introduction; Declaring a record; Manipulating a record—the WITH statement; Arrays of records; The set data type; Set operations; Searching an array of records; Common programming errors; Summary; Programming problems. *REPEAT and GOTO Statements, Nested Structures and Recursion*—Introduction; REPEAT-UNTIL loop; Nested loops; Sorting an array; The GOTO and EXIT statements; Solving a larger problem; Testing a program system; Recursion; Common programming errors; Summary; Programming problems. *Hierarchical Records and Files*—Introduction; Hierarchical records; Record variants; TEXT and INTERACTIVE files; User-defined file types; File update and merge; Common programming errors; Summary; Programming problems. *Multidimensional Arrays*—Introduction; Declaration of multidimensional arrays; Manipulation of multidimensional arrays; Room scheduling; Introduction to computer art: drawing block letters; Common programming errors; Summary; Programming problems. *Pointer Variables and Dynamic Data Structures*—Introduction; The NEW statement and pointers; Building linked data structures; Deleting a node; List insertion; Multiple-linked lists and trees; Common programming errors; Summary; Programming problems. *Appendices—1.* Differences Between UCSD Pascal and Standard Pascal; *2.* Reserved Words: Standard Identifiers and Operators; *3.* Using UCSD Pascal; *4.* Pascal Syntax Diagrams. *Index of Program Style Displays. Index of Programs, Procedures and Functions. Answers to Selected Exercises. Index.*

Don't! (Or How to Care for Your Computer) by Rodnay Zaks. Sybex (Berkeley, CA), 1981, 224 pages, 6 × 9 inches, paperback.
ISBN: 0-89588-065-2 $11.95

An explanation of how to handle and maintain all components of a computer system: the CRT display, the diskettes, the printer, the magnetic tapes. Contains cartoons and photographs.

CONTENTS: *Caring For Your Computer*—Introduction; Why Bother? Are Computers Reliable? Is The Computer Foolproof? Controlling Your Emotions; The Time Bomb; The Pointed Index Syndrome; It Is So Simple. *The Computer System*—Introduction; The Monitor; The Memory; The Operating System; The Files; The Mass Storage Media; The CRT Terminal; The Printer; Summary. *Floppy Disks*—For The Home Computer User; Introduction; Understanding Your Diskette; Handling the Diskette; Using The Diskette; Backing-Up; Labeling; Storing Diskettes; Environment; Transporting Diskettes; Preventive Maintenance; Disk Failures; Floppy Disk Summary. *Hard Disks*—For the Home Computer User; Introduction; Understanding Your Disk; Using Hard Disks; The Main DOs and DON'Ts — A Summary. *The Computer*—For The Home Computer User; Introduction; Understanding Your Computer; Operating The Computer; Inside The Computer; Computer Summary. *The CRT Terminal*—For The Home Computer User; Introduction; The Operator's Working Environment; Environmental Requirements; Using The CRT; External Video Monitor Or TV; CRT Summary. *The Printer*—For The Home Computer User; Introduction; Types of Printers; Installing The Printer; Connecting The Printer; The Environment; Maintenance; Printer Failures; Supplies; Printer Summary. *The Tape Units*—For The Home Computer User; Introduction; Handling Tapes; Environment And Storage; Shipping Tapes; Tape Problems; Maintenance; Tape Units Summary. *The Computer Room*—For The Home Computer User; Introduction; Floor Planning; Electrical Power; The Environment; Furniture; Fire Protection; Procedures; Summary. *Software*—For The Home Computer User; Introduction; Software Requirements; Workspace Requirements; Software Facilities; Software Maintenance; Software Procedures; Hardware Changes; Software Changes; Summary. *Documentation*—For The Home Computer User; Introduction; Hardware Documentation; Software Documentation; Record Of Changes; Summary. *Security*—Introduction; Erecting Barriers; Protecting Forms; Securing The Site; Encryption; Audit Trails; Computer Theft; Summary Of Security Procedures. *Help*—Introduction; The Two Types of Maintenance; Securing Maintenance Services; When It Doesn't Work; Summary; Conclusion. *Appendix A*—Tape and Disk Manufacturers. *Appendix B*—Useful Addresses. *References. Index. Library.*

MICRO

POOL 1.5
ATARI OWNERS,
PRICE: $34.95
• High resolution graphics-Life like motion.
• Instant replay on any shot.
• Superfine aiming.
• Pool room sound effects
• Choice of 4 popular games
See your local computer dealer or order directly from IDSI. Specify ATARI or APPLE.
IDSI
ATARI is the registered trademark of ATARI Inc.
APPLE II is the registered trademark of APPLE Computer Inc.
POOL 1.5 is trademark of IDSI.

MICRO

From Here to Atari

By James Capparell

Character sets, display codes, ATASCII codes, and keyboard codes are the subject of this month's column. This information will help you understand how a character appears on your T.V. screen. I will show you what the Atari character set is, where it resides in ROM, and how to access the character set. I'll also describe the three codes used to refer to the character set. Program 1 will print the keyboard code, ASCII character, and display code for any given key. Program 2 will show you how to place characters on a graphics 8 high-resolution screen. However, before we get that far we need to know what happens when a key is pressed on the keyboard.

When you press any key, an IRQ interrupt is generated. (For more on this, see my column in the January '82 MICRO.) The vector for IRQ is at memory location $216, $217 called VIMIRQ in the documentation. This vector points to $E6F6, the entry point for the IRQ Interrupt Service Routine (ISR). This service routine performs the following functions:

1. Saves system registers.

2. Determines cause of interrupt by polling status register bits. IRQEN at $D20E is interrogated for this purpose. See table 1 for IRQEN bit translations.

3. Jumps through the appropriate vector to the ISR. The ISR performs the necessary housekeeping associated with the interrupting source.

For the sake of our discussion, assume a key has been pressed. This causes an IRQ. Once it has been established that it was a keyboard-caused IRQ, a jump is made through locations $208, $209 called VKEYBD. This location contains $FFBE, the start-of-keyboard service routine. This routine performs the following functions:

1. Processes debounce. Bounce is associated with the mechanical vibration caused by key closure. This bounce can appear to the system as several keystrokes instead of just one. A software delay of 20 msec is sufficient to allow the vibration to dampen. A counter for this purpose is established at this point.

2. Starts/Stops(cntrl-1) processing. SSFLAG at location $022F is set when the control and "1" keys are pressed simultaneously. This is the feature that allows you to start and stop listings or your favorite game.

3. Saves the keyboard code in locations $2FC and $2F2, called CH and CH1, respectively. This code is to be differentiated from ATASCII or the display codes.

Table 1

Address $D20E, known as IRQEN (interrupt request enable), is interrogated whenever an IRQ interrupt occurs.

Bit 7	=	Break key interrupt
Bit 6	=	Other key interrupt
Bit 5	=	Serial input data ready interrupt
Bit 4	=	Serial output data needed
Bit 3	=	Serial output transmission finished
Bit 2	=	Timer 4 decremented to 0
Bit 1	=	Timer 2 decremented to 0
Bit 0	=	Timer 1 decremented to 0

Listing 1

```
5 REM ** PROGRAM 1 **
6 REM PRESS ON ANY KEY WAIT FOR A COUPLE OF SECONDS
7 REM THE KEYBOARD CODE, CHARACTER, AND THE CHARACTER'S
8 REM DISPLAY CODE ARE PRINTED
10 OFFSET=6
20 DMEM=PEEK(88)+PEEK(89)*256:REM FIND DISPLAY MEMORY
30 A=PEEK(764):IF A>99 THEN OFFSET=7:IF A<10 THEN OFFSET=5
40 IF A<>255 THEN ? A;"  ";CHR$(A);"  ";:? PEEK(DMEM+OFFSET)
50 OFFSET=6
60 GOTO 20
```

Listing 2

```
5 REM ** PROGRAM 2 **
6 REM PUT TEXT ON A GRAPHICS 8 SCREEN
7 REM CHANGE X, Y SEE WHAT HAPPENS
10 DIM OUT$(15),CNVRT$(1)
15 OUT$="ATARI 800":REM MESSAGE
20 CHBAS=57344:REM START OF CHARACTER SET
22 SPACE=2
25 X=12:Y=85:REM HORZ. VERT. OFFSETS
30 GRAPHICS 8+32
35 DMEM=PEEK(88)+PEEK(89)*256:REM START OF DISPLAY MEMORY
40 DMEM2=DMEM+X+(Y*40):REM OFFSET TO SCREEN CENTER
45 FOR I=1 TO LEN(OUT$):REM MOVE MESSAGE
50 CNVRT$=OUT$(I,1):GOSUB 1000
55 CHAR=CHBAS+X*8:REM GET CHARACTER DATA
60 FOR BYTE=0 TO 7
65 POKE DMEM2+BYTE*40,PEEK(CHAR+BYTE)
70 NEXT BYTE
75 DMEM2=DMEM2+SPACE
80 NEXT I
85 STOP
900 REM SUBROUTINE CONVERTS ATASCII INTO DISPLAY CODES
910 REM DISPLAY CODE USED AS INDEX INTO CHARACTER SET IN ROM
1000 X=ASC(CNVRT$)
1010 IF X>127 THEN X=X-128:REM ELIMINATE REVERSED CHAR.
1020 IF X>31 AND X<96 THEN X=X-32:RETURN
1030 IF X<32 THEN X=X+64
1040 RETURN
```

From Here to Atari

(Continued)

4. Sets attract mode flag at location \$4D. This prevents color rotation, which normally occurs after nine minutes of keyboard inactivity. If you choose to disenable color rotation, be aware that prolonged operation without rotation could damage your picture tube's phosphor.

5. Sets location \$22B, called SRTIMR, to \$30. This is the auto-repeat timer and is used by Stage 2 Vblank routines to auto-repeat any key that is held down longer than ½ second. Stage 2 Vblank processing also decrements the debounce counter and updates the auto-repeat timer every 1/60 second.

After a key has been processed through the keyboard interrupt routines and Vblank, the resident keyboard handler takes over. This handler is part of the versatile Central Input/Output CIO facility. Most of what goes on here is very involved and the interested reader is advised to go to the operating system listing to follow the flow. These listings are available from Atari (ask for manual C016579). The one function that CIO performs that we need to understand is code translation.

Figure 1

```
.  .  .  .  .  .  .  .   = $00
.  .  .  x  x  .  .  .   = $18
.  .  x  x  x  x  .  .   = $3C
.  x  x  .  .  x  x  .   = $66
.  x  x  .  .  x  x  .   = $66
.  x  x  x  x  x  x  .   = $7E
.  x  x  .  .  x  x  .   = $66
.  .  .  .  .  .  .  .   = $00
```

The hex values are those found in ROM locations 57608 - 57615. These are the stored values representing the letter A.

If you read the manuals, you know there are many references to ATASCII code. Atari ASCII or ATASCII is Atari's version of the American Standard Code for Information Interchange. ASCII is an industry-standard description of how 26 letters of the alphabet, numbers, special punctuation, and some special characters can be represented in eight bits. Since there are 256 combinations available in eight bits, this leaves many combinations unused in the normal ASCII. The Atari, however, uses them all since it can display special graphics characters, inverted characters, and normal characters.

A universally accepted code — e.g. ASCII — is essential for devices to communicate properly with one another and with us. If the serial bit stream 01000001 is sent to any printer which recognizes ASCII, it will print the capital letter "A". Look at Appendix C-1 in your BASIC reference manual to see the entire ATASCII code and characters.

ATASCII is included in our machines to be compatible with peripheral devices. The Atari display code for each character is different from ATASCII. The display code is used to access the actual data that forms a character. It is all this data that is collectively referred to as a character set. The entire character set is stored in ROM starting at page address \$E0 (that's 57344 decimal). This character set is simply a string of bytes describing the shapes of individual characters.

Each character requires eight bytes, and is formed in an 8 × 8 grid. See figure 1. In order to access the appropriate eight bytes it is necessary to know the display codes of the character set. Program 1 is designed to tell you what a given code is for any key pressed on the keyboard, and will also work for shifted or controlled keys. Once we have found the data for the character we want, we can use that data. Look at program 2 to see how we moved letters, byte-by-byte, and stacked these bytes one on the top of the other to display characters in graphics mode 8.

The data stored at any location within a character set is arbitrary. Suppose when we go look for the string of bytes that normally is an "A" some other data is stored there. It would only be possible for different data to be there if the character set had been moved to RAM. Atari gave us another pointer called CHBAS \$D409. This location tells the O.S. where the first page of the character set data is. Normally residing in ROM, it can be moved to RAM. New data replaces old, and the pointer CHBASE can be changed to reflect the new location of the data. It is in this way that the letter "A" can be replaced by any pattern that will fit into the normal 8 × 8 grid. This process, known as redefining character sets, requires a few basic steps.

1. The new characters must be designed. Recall that each character must fit into an 8 × 8 grid. Then these byte values must be moved to an appropriate place reserved in memory just for this purpose.

2. ANTIC must be informed of where the redefined character set is in RAM. The character set must be on a 1K boundary, and CHBAS, location \$2F4, must be changed to point to the page address of the new character set.

Using some of these ideas, you could change the delay before a key repeats, redefine the keys on the console, use the keyboard vector to trap certain keys and give them special meaning. Well, you get the idea — it's completely flexible.

The author may be contacted at 297 Missouri St., San Francisco, CA 94107.

MICRO

MICRO The Single Life

By Brad Rinehart

In the single hoard world, where virtually each machine can be slightly different, we need software that can adapt itself to several configurations. In contrast, other personal computer people do not need to be quite as careful: their machines are virtually the same. They do not have to worry ahout dealing with equipment produced by different manufacturers and configured together to build a system. But they also do not have the luxury of custom-tailoring their systems to the wide variety of equipment available.

As I'm sure you are aware, there are at least a hundred different terminals available. Some offer video, some hard copy only; others combine both. It is difficult to accommodate all the different possibilities, but the people at HDE have taken care of part of the job for you.

HDE Disk BASIC employs a feature called the personality module. The user has access to several locations contained in this module. These locations can be used to define backspace, hackarrow, escape, edit inserts, cancel functions, and CLEAR SCREEN. There is a token (or command) in HDE Disk BASIC called CLS. CLS stands for CLear Screen. The personality module provides a function which is used to define the character sequence that is sent for the CLS command. Therefore, in your BASIC programs, when you want to clear the screen use a CLS command. For hardcopy terminal users, CLS transmits seven nulls if no characters are defined for it.

HDE Disk BASIC has a feature called LOADD, or load data. The full syntax for the load data command is:

```
LOADD "FILE NAME", DRIVE NO.
```

The drive numher is optional; drive number two is the default. This command allows us to set up a common data file. In this file we can predefine any legal variables to any legal amount. This includes strings, arrays, numeric variables, integers, etc. Once they are defined in this common data file, they can he loaded to memory from disk using the LOADD command. This accomplishes two things. First it loads our predefined variables, and second it ERASES all other variables. Instant free memory! But how is this used?

You enter HDE Disk BASIC the same way the old KIM BASIC was entered from FODS: enter the FODS command, BAS (RETURN). This loads BASIC from the disk and initializes it. Or you can enter BAS. (note the period) which loads and intializes BASIC. When BASIC "sees" the period following its name, it looks to the system disk (either drive #1 or drive #0) for a BASIC program called MENU. If BASIC finds MENU, it executes it. If you change the first three locations in FODS to NOPS (EA, EA, EA), FODS performs this from the hoot strap. Instant auto start! Of course BASIC, its supporting routines, and your MENU program, must all he on your FODS system disk for this to happen.

The statement IF# (see figure 1) allows BASIC to "look" at the disk and determine if a program or file exists. In our example we are looking for a file named "COMMN". By adding the ,1 to the statement, we specify that we want HDE BASIC to look for the file on drive #1. We could have substituted a string variahle for the file name, and a numeric variahle for the drive number.

If the file exists, the IF# statement will be true. Therefore, the LOADD "COMMN",1 statement will be executed. If the file does not exist, the IF# statement will be false, and the ELSE statement will be executed, CHAINing the program SETUP from drive 1.

Let's assume that this is the first time this program has been run, our common data file does not yet exist, and control will he passed to our program SETUP. The next step is to determine what type of terminal is attached to the system. First I suggest you set up a menu which lists the types of terminals supported by the software. You may also want to add a menu selection which would allow the user to enter information for nonsupported terminals. For now, we'll assume the system is equipped with one of the terminals supported hy the software.

Based on the user's entry to our menu, we would GOSUB to a routine similar to figure 2. From the REMarks you can see that this routine sets up cursor controls for the Hazeltine 1400 and 1500 series of terminals. We would normally have similar routines for all other terminals supported by our software. Then, if we want to do a HOME CURSOR, we simply PRINT CU$; from BASIC. Iregardless of which terminal is attached to the system, PRINT CU$ will position the cursor to HOME.

Once all of the proper variables have been defined, it is necessary to have the software "remember" them. The SAVED, or SAVE Data command, can be used to write all the current variables to disk. For example, the command SAVED"COMMN",1 will save the data to drive 1, under the name "COMMN". Then, whenever we need to load the data from disk, we use the LOADD"COMMN",1 command.

Addressing the cursor to an X, Y coordinate on the screen is a bit more complex. In figure 3, you will find an appropriate routine. Although no two terminals are alike, most require a LEAD IN character which tells the terminal that a command follows.

In figure 2 we defined a string called AD$, which is our LEAD IN. It is normally followed by the row and column to which we address the cursor. To use the subroutine in figure 3, we first set up the variahles R and C to the row and column we are addressing. Then we perform a GOSUB 1000.

The variable TT, or terminal type, was set up in our SETUP program when the user entered the terminal type for the system. TT, along with our cursor control characters, was "remembered" by the SAVED"COMMN",1 command. The variahles '01, 02, and 03' were also defined in the SETUP routine. They are used to define any standard "offset" that may have to be added to the row and column for use with a particular terminal. Using a routine similar to this one eliminates the problem of rewriting the software for different terminals. It may take a little extra time to set this routine up in your program, but it will he well worth it in the long run.

Note that, if you address the cursor, print some information, and do a GOSUB 1010, the cursor will be repositioned to the beginning of the information just printed. This point is very useful when entering information into screen masks or forms. You can print a line of stars (****) signifying the length of the information to he input, and then position the cursor to the heginning of the stars.

Common data files have other uses as well. HDE Disk BASIC currently supports from one to three five-inch or eight-inch disk drives. In addition, these drives may be either single- or double-sided. Using the common data file technique, we can assign variables which define these parameters.

Where HDE Disk BASIC is concerned, all single-sided drive disk systems operate with their system disk originally assigned as either drive number zero (0), or one (1). Therefore, if our BASIC programs are stored on the system disk, we may load them by specifying either LOAD 'PGM NAME'',0 or LOAD ''PGM NAME'',1. In the case of double-sided drives, the system disk is always drive zero (0). To load programs from the system disk, it is necessary to use the command LOAD ''PGM NAME'',0. From this example we see that it is best to specify the system drive in HDE Disk BASIC as drive zero (0) because this conforms to both the double- and single-sided drive standards.

You'll find it beneficial to predefine variables such as a system password, the maximum number of records allowed in a file, and the default system device drive name (as for a printer, modem, etc.). If the user wants to upgrade his system he only needs to delete the common data file from the disk, rerun the MENU program, and redefine the proper variables.

Please address correspondence to: 1500 Stanton Street, York, PA 17404.

Figure 1

```
90 REM SEE IF COMMON DATA FILE ON DISK
100 IF#"COMMN",1 THEN LOADD"COMMN",1. ELCE CHAIN "SETUP",1
110 REM
```

Figure 2

```
60000 REM CURSOR CONTROL SUBS FOR HAZELTINE 1400, 1500
60005 REM
60010 LE%=126.REM LEAD IN
60020 CL$=CHR$(LE%)+CHR$(28):REM CLEAR SCREEN
60025 CU$=CHR$(LE%)+CHR$(18):REM HOME
60040 UP$=CHR$(LE%)+CHR$(12).REM UP CURSOR
60045 DO$=CHR$(LE%)+CHR$(11).REM DOWN CURSOR
60050 AD$=CHR$(LE%)+CHR$(17).REM ADDRESS CURSOR
60055 CE$=CHR$(LE%)+CHR$(15):REM CLEAR TO END OF LINE
60060 CP$=CHR$(LE%)+CHR$(24):REM CLEAR TO END OF PAGE
60065 UL$=CHR$(LE%)+CHR$(6):REM KEYBOARD UNLOCK
60070 LK$=CHR$(LE%)+CHR$(21).REM LOCK KEYBOARD
60090 01=32:02=31:03=96:TT$="HAZ"+"L":GOTO60600
60095 REM
```

Figure 3

```
980 REM ADDRESS CURSOR SUBROUTINE
990 REM
1000 GOTO1000+TT
1001 R=R+01:IFC<02THENC=C+03.SWAPR,C:GOTO1010.ELSESWAPR,C.GOTO1010
1002 R=R+01:C=C+01:GOTO1010:REM Lear Seigler ADM 3/ADM 5
1003 R=R+01:C=C+01:GOTO1010.REM ADDS Regent
1004 R=R+WN:C=C+WN:GOTO1015:REM DEC VT100/VT103
1010 IFTT<FRTHENPRINTAD$;CHR$(R);CHR$(C),:POKE22,ZR:RETURN
1012 REM This line to handle DEC VT100 and VT103
1015 PRINTAD$,RIGHT$(STR$(R),LEN(STR$(R))-1);",".
1020 PRINTRIGHT$(STR$(C),LEN(STR$(C))-1);"H";:POKE22,0.RETURN
1070 REM
```

MICRO Hardware Catalog

Name: **TRS-80 Color Computer Learning Lab (26-3153)**
System: TRS-80 Color Computer
Memory: 4K, 16K, 32K
Language: Color BASIC
Hardware: Cassette
Description: A new self-instruction system that teaches user how to program in Color BASIC. Allows student to develop gradually through writing and editing longer, more complex programs. Example programs are practical and can be used for educational, family and personal purposes. The lab is divided into three sections: introduction to the computer; programming the computer; programming guides and tools to make programming easier, faster and more fun. The lessons take full advantage of color graphics and sound available from the TRS-80 Color Computer.
Price: $49.95 includes eight program cassettes and 30-lesson text
Author: Radio Shack
Available:
Radio Shack

Name: **Format ROM**
System: Apple II or Apple II Plus
Hardware: M.C.'s ROMPLUS or Andromeda's ROMBoard
Description: *Format ROM* will give you word processing and print using power for your Applesoft programs. Print Statement Formatting, a word processing system, looks at print statements in your program and makes sure they are properly formatted to your predefined requirements before Applesoft outputs them to your monitor or printer. The *Format ROM* will right and left justify, indent or outdent paragraphs, center text, pause, skip lines, redefine characters, and more. Print using commands will tabulate, right justify, line up decimal places, pad the right and left side of a number with any predefined character, insert commas every third digit from left of decimal place, can be used within formulas and equations, has overflow capabilities, and more.
Price: $49.95
Available:
Soft CTRL Systems
P.O. Box 599
West Milford, NJ 07480
(201) 728-8750

Name: **Percom M65/50 Adapter**
System: AIM 65, KIM, SYM
Description: Interface adapter board which allows AIM, KIM and SYM computer owners to expand their systems with standard System-50 (SS-50) modules. This gives the owner the advantage of being able to upgrade to disk storage, a CRT display and other devices using low-cost off-the-shelf SS-50 modules.
Price: $89.95 includes adapter, SS-50 motherboard
Available:
Percom Data Company, Inc.
11220 Pagemill Rd.
Dallas, TX 75243
1-800-527-1222
for orders

Name: **ETI²**
System: Any
Memory: 2K
Hardware: Z80
Description: The Mediamix *ETI²* is an intelligent interface that connects the IBM Electronic Typewriter Models 50, 60, 75 or 175 to any computer. Available in Serial RS-232C or Centronics-compatible parallel versions, the Mediamix *ETI²* adds RO printing capability to the typewriter as well as provides for total access to all of the special functions of the typewriter through the computer.
Price: $495 Parallel, $595.00 Serial, includes cable and connector, power supply, and full documentation.
Available:
Mediamix
P.O. Box 67B57
Los Angeles, CA 90067
and selected dealers

Name: **Micromodule 16 (M68MM16-1; M68MM16-2; M68MM16-3)**
Memory: 16K to 64K bytes of dynamic RAM
Hardware: Chassis and card cages (available from Motorola)
Description: *Micromodule 16* provides RAM, ROM, I/O and timer expansion for the Motorola Micromodule™ series of 8-bit monoboard microcomputers. It incorporates 2K bytes of static RAM which can be backed up with an external battery and power-fail detect circuit, and mounts an additional four 24-pin sockets in which the user may install his choice of 1K, 2K, 4K or 8K EPROMS, MOS or Bipolar PROMS, mask ROMs or pin-compatible RAMs for up to 32K of additional memory.
Price: $575 each
Available:
Motorola Systems distributors

Name: **RAMPLUS +**
Description: Dual 16K RAM card for the Apple II. Two banks of 16K selectable RAM expand the Apple to 80K of available memory. The second bank of 16K RAM is controlled by user-supplied software. Hardware and/or software selection of each 16K bank of RAM is controlled by the user. The card also provides its own refresh circuitry. Just install it in any I/O slot; no bothersome cables to connect or IC's to remove from the Apple motherboard.
Price: $189.00 includes 16K of installed RAM
Available:
Mountain Computer Inc.
300 El Pueblo Road
Scotts Valley, CA 95066
(408) 438-6650

Name: **Dual Thermometer**
System: Apple II
Memory: 48K
Language: Applesoft
Hardware: One disk drive (13- or 16-sector) required; Printer and Clock Card optional
Description: Use the *Dual Thermometer* to measure two temperatures almost anywhere. The software lets you display temperatures on the screen — maximum, minimum, the difference, and time. Store this data on the disk or printer. Set the limit for an alarm to sound. Range: −55 to 125°C. Accuracy ±0.4°C from −20 to 50°C. (−4 to 122°F), ±1°C outside that range. Probes can be extended 500 feet.
Price: $260.00 includes interface card, two 10-foot temperature probes, diskette and 83-page manual.
Available:
Strawberry Tree
949 Cascade Drive
Sunnyvale, CA 94087
(408) 736-3083
or your dealer

Name: **MDM-1**
System: VIC-20
Hardware: Commodore Vic
Description: The *MDM-1* is a modem driver module that plugs directly into the user port of the Commodore VIC microcomputer. It has two additional serial ports for connecting a modem and a serial printer. It does not require any external power. A terminal program supplied with the unit permits telephone line connections to large timesharing computers, and to various computer networks, such as Source, Compu-Serve, Dow Jones, etc. There are two indicators on the *MDM-1* that illustrate the transmissions to and from the VIC. These transmissions are simultaneously recorded on the VIC TV screen and the serial printer.
Price: $59.00 plus $3.00 shipping
Available:
RVR Systems
P.O. Box 265
Dewitt, NY 13214

Name: **Signalman**
Hardware: Modem Direct Connect
Description: 300 baud direct connect modem includes RS-232C cable, tone-audible carrier detect, automatic mode selection, and tone-audible battery warning (when low). Talk data switch optional. D.C. wall transformer. Size 8 × 4 × 1.; weight 10 lbs.
Price: $99.00
Available:
Anchor Automation Inc.
16130 Valerio Ave.
Van Nuys, CA 91406
(213) 997-6493

MICRO

Apple II

sensational software

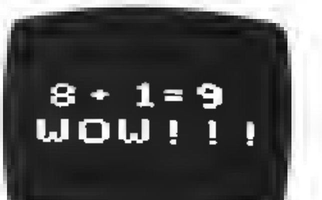
creative computing software

CAI Programs Vol I

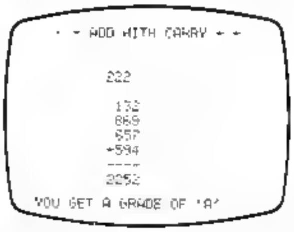
Cassette CS-4201 $11.95 — Requires 16K Apple II or Apple II Plus

U.S. Map Identify states and their capitals.

Spelling. Study aid with your list of trouble-some words.



Math Drill Arithmetic drill and practice with large or small display.



Add With Carry Drill and practice on sums requiring numbers to be carried

Ecology Simulations - I

Disk CS-4706, $24.95 — Requires 48K Applesoft in ROM or Apple II Plus

Sterl
STERL allows you to investigate the effectiveness of two different methods of pest control—the use of pesticides and the release of sterile males into a screw-worm fly population. The concept of a more environmentally sound approach versus traditional chemical methods is introduced. In addition, STERL demonstrates the effectiveness of an integrated approach over either alternative by itself.

Pop
The POP series of models examines three different methods of population projection, including exponential, S-shaped or logistical, and logistical with low density effects. At the same time the programs introduce the concept of successive refinement of a model, since each POP model adds more details than the previous one

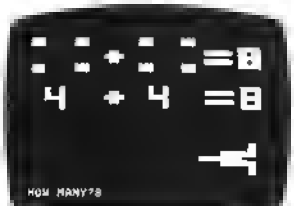
Tag
TAG simulates the tagging and recovery method that is used by scientists to estimate animal populations. You attempt to estimate the bass population in a warm-water, bass-bluegill farm pond. Tagged fish are released in the pond and samples are recovered at timed intervals. By presenting a detailed simulation of real sampling by "tagging and recovery," TAG helps you to understand this process

Buffalo
BUFFALO simulates the yearly cycle of buffalo population growth and decline, and allows you to investigate the effects of different heard management policies. Simulations such as BUFFALO allow you to explore "what if" questions and experiment with approaches that might be disatrous in real life.

CAI Programs Vol II

Cassette CS-4202 $11.95 — Requires 16K Apple II or Apple II Plus

European Map Identify countries and their capitals.



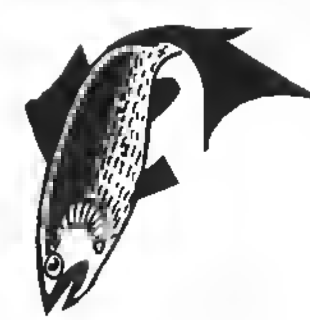
Meteor Math Learn math skills by destroying menacing meteors

Music Composing Aid Make and play your own music on the Apple. No additional hardware required. Includes a sample from Bach's Tocatta & Fugue in D minor

Ecology Simulations - II

Disk CS-4707 $24.95 — Requires 48K Applesoft in ROM or Apple II Plus

Pollute
POLLUTE focuses on one part of the water pollution problem; the accumulation of certain waste materials in waterways and their effect on dissolved oxygen levels in the water. You can use the computer to investigate the effects of different variables such as the body of water, temperature, and the rate of dumping waste material. Various types of primary and secondary waste treatment, as well as the impact of scientific and economic decisions can be examined



Rats
In RATS, you play the role of a Health Department official devising an effective, pratical plan to control rats. The plan may combine the use of sanitation and slow kill and quick kill poisons to eliminate a rat population. It is also possible to change the initial population size, growth rate, and whether the simulation will take place in an apartment building or an entire city

Malaria
With MALARIA, you are a Health Official trying to control a malaria epidemic while taking into account financial considerations in setting up a program. The budgered use of field hospitals, drugs for the ill, three types of pesticides, and preventative medication, must be properly combined for an effective control program

Diet
DIET is designed to explore the effect of four basic substances, protein, lipids, calories and carbohydrates, on your diet. You enter a list of the types and amounts of food eaten in a typical day, as well as your age, weight, sex, health and a physical activity factor. DIET is particularly valuable in indicating how a diet can be changed to raise or lower body weights and provide proper nutrition

CAI Programs I and II

Disk CS-4701, $24.95
Requires 32K Integer Basic

This disk contains all 7 programs from cassettes CS-4201 and CS-4202

Note The ecology simulations programs are not available on cassette

Stock & Options Analysis

Disk CS-4801, $99.95
Requires 32K Applesoft or Apple II Plus

This is a comprehensive set of four programs for the investment strategy of hedging listed options against common stocks. A complete description is in the TRS-80 section. Available August 1981

Order Today

To order any of these software packages, send payment plus $2.00 postage and handling per order to Creative Computing, Morris Plains, NJ 07950 Attn: Cleo

Visa, MasterCard and American Express orders may be called in toll-free

Order today at no risk. If you are not completely satisfied your money will be promptly and courteously refunded

Creative Computing Software
Morris Plains, NJ 07950
Toll-free **800-631-8112**
In NJ 201-540-0445

creative computing software

Apple is the registered trademark of Apple Computer Inc

MICRO Software Catalog

Name: **Star Zap**
System: OSI
Memory: 8K
Language: BASIC
Hardware: Cassette
Description: *Star Zap* is a high speed re-creation of the popular arcade game. You must defend your starbase against the aliens who attack from all four quadrants at once! Only fast reflexes can save you! Includes machine code for sound on all C1Ps and Superboards, color and sound on C4Ps.
Price: $9.95
Author: John Wilson
Available:
Pretzelland Software
2005 Whittaker Rd.
Ypsilanti, MI 48197
(313) 483-7358

Name: **Universe**
System: OSI C1, C2-4
Memory: 8K tape
20K disk
Language: Machine
Description: Pilot your space ship across the surface of Arcton IV while engaging the enemy rockets and dodging meteorites. Can you maneuver through the mountains without being blown up into a thousand pieces? If you can, then be prepared for more action than you thought possible on your OSI computer. You can use your keyboard or a joystick to control your ship.
Price: $14.95 includes 5¼" disk or tape and instructions
Author: Dave Pompea
Available:
DMP Systems
319 Hampton Blvd.
Rochester, NY 14612

Name: **The Vaults of Zurich**
System: PET, Atari
Memory: 16K PET
24K Atari
Language: BASIC
Hardware: Cassette or diskette
Description: Zurich is the banking capital of the world. The rich and powerful deposit their wealth in its famed impregnable vaults. But you, as a master thief, have dared to undertake the boldest heist of the century. You will journey down a maze of corridors and vaults, eluding the most sophisticated security system in the world. Your goal is to reach the Chairman's Chamber to steal the most treasured possession of all: The OPEC Oil Deeds!
Price: $21.95 cassette
$25.95 diskette
Author: Felix and Greg Herlihy
Available:
Artworx Software Company
150 N. Main Street
Fairport, NY 14450
800-828-6573 or
(716) 425-2833

Name: **The Accountant Finance Data Base System**
System: Apple II, Apple II Plus with DOS 3.3 and Applesoft in ROM
Memory: 48K
Language: Applesoft BASIC
Hardware: Single or dual drives
Description: A double entry accounting system that prompts the user for the account(s) that are to be increased and/or decreased. The system permits the user to define his own account names and tax codes. Ad hoc queries and daily reports feature a natural dialogue. A VisiCalc interface is available.
Price: $99.95
Includes user manual, demo database and tutorial
Author: Ernest H. Forman
Available:
Decision Support Software
1438 Ironwood Drive
McLean, VA 22101
(703) 241-8316

Name: **Business Planner**
System: Apple II
Memory: 48K
Language: Pascal
Description: *Business Planner* is a modeling package for entrepreneurs planning to start or expand a new business. Designed to help develop viable business plans, the program groups labor, equipment and other costs into income-related projects. Projects are combined into a model which generates graphical projections and estimated financial statements. "What If" scenarios help you plan for the future and respond appropriately to changing demands.
Price: $290.00
Includes three diskettes and manual
Available:
Duosoft Corporation
Box 1827
Champaign, IL 61820
(217) 356-7542

Name: **Investment Decisions**
System: Apple II, Apple II Plus, Apple III
Memory: 48K (Apple II)
48K (Apple II Plus)
96K (Apple III)
Language: Applesoft (Apple II and Apple II Plus); Business BASIC (Apple III)
Hardware: Apple II with firmware card
Description: Package components: loan schedules, savings schedules, annuity schedules, depreciation schedules, amortization schedules, APR schedules, payback method, net present value, internal rate of return, profitability index.
Price: $100.00
Includes disk, documentation and run instructions
Author: J.L. Campbell
Available:
Mesa Research, Inc.
Rt. #1, Box 1456A
Waco, Texas 76710

Name: **Capitalization**
System: Apple II Plus
Memory: 48K
Language: Applesoft
Description: This 2-disk system provides for practice and testing on the application of the major rules of capitalization. The practice disk presents a rule followed by up to 20 randomly presented sentences which provide practice on the rule. The test disk measures the student's ability to correctly apply rules of capitalization. It may be used as a pre-test or post-test. The management system gives immediate feedback to the student and stores records of each student's test results for later review by the teacher. Results may be printed or viewed on the screen. The teacher has the ability to modify or add new materials to either disk. The lessons using upper/lower case letters are appropriate for levels 3-8.
Price: $49.95
Includes 2 disks plus documentation
Author: Hartley Staff
Available:
Hartley Courseware, Inc.
Box 431
Dimondale, MI 48821
(616) 942-8987

Name: **Cave Hunter**
System: TRS-80 Color Computer
Memory: 16K
Language: Machine
Hardware: Joysticks
Description: A fast-paced arcade game using Hi-Res graphics, sensational colors and a variety of unique sounds. Single or multiple players. Maneuver your way to the bottom of a spooky old cave to retrieve the treasures. It's not so easy! Passages lead in all directions and angry cave creatures pursue you relentlessly.
Price: $24.95
Includes cassette, directions and ppd shipping
Author: Ron Krebs
Available:
Mark Data Products
23802 Barquilla
Mission Viejo, CA 92691

Name: **Moment of Inertia & Element of Triangle**
System: Apple II, Apple II Plus
Memory: 32K with DOS 3.3 or 3.2 with FP installed
Language: Applesoft BASIC
Hardware: DOS 3.2 or 3.3 with controller card
Description: The *Moment of Inertia* contains 56 physical formulas for 22 various bodies of mass. It calculates dimension, choice of mass or inertia on selected axis. This program is intended for engineers who never considered these important factors. The *Element of Triangle* program contains three major triangles (right, equilateral and general); 23 formulas; calculates sides, angles, altitude, area and radius of inscribed circle simultaneously to find force and directions. Both programs are packed in single diskette so they may be

used interactively. The menu program will display all formulas and the definition program will define all details. Both programs utilize touch-key selection input system, eliminating use of return key, but recognizes characters or initial (abbreviation) of known elements to find the formula and provide missing variables. Instructions include more than 100 commonly used industrial materials.

Price: $40.00
Includes both programs on diskette

Available:
American Avicultural Art & Science, Inc.
3268 Watson Rd.
St. Louis, MO 63139
(314) 645-4431

Name: **COLORFORTH**
System: TRS-80 Color Computer
Memory: 16K minimum
Language: FORTH
Hardware: Cassette or Radio Shack Color Disk System

Description: *COLORFORTH* is a special implementation of fig-FORTH for the TRS-80 Color Computer. This program requires a minimum of 16K, but does not require Extended BASIC. (Extended BASIC *is* required by the Radio Shack Color Disk operating sytem.) Includes an Editor and CSAVEM command normally not available without Extended BASIC, printer commands and much more! Write or call today. Visa and MasterCard accepted.

Price: $49.95 ppd. Texas residents add 5% sales tax
Includes cassette and disk versions and 31-page manual

Available:
Armadillo Int'l. Software
P.O. Box 7661
Austin, TX 78712
(512) 459-7325

Name: **HSD Regress**
System: Apple II or Apple II Plus, DOS 3.2 or 3.3
Memory: 48K
Language: Applesoft
Hardware: Optional printer, serial or parallel interface, Silentype

Description: *HSD Regress* is a menu-driven multiple regression package which accepts up to 25 variables of 300 data points each. Data can be entered from keyboard or disk. All data can be reviewed and edited, transformed numerically, and stored on disk. Multiple regression analysis can be performed on all variables input, or on any subset of variables, in any order. Output includes correlation matrix, predicted and residual scores, matrix inverse, semi-partial correlations, coefficient weights and p-values.

Price: $99.95
Includes disk, complete documentation, imprinted 3-ring binder

Author: Stephen Madigan
Virginia Lawrence

Available:
Human Systems Dynamics
9249 Reseda Blvd. Suite 107
Northridge, CA 91324
(213) 993-8536
or selected computer stores

Name: **Waterloo microAPL**
Hardware: Commodore SuperPET, Volker-Craig 2900, 3900, 4900, Northern Digital microWAT

Description: *Waterloo microAPL* is intended to be a complete and faithful implementation of the IBM/ACM standard for APL with respect to the syntax and semantics of APL statements, operators and primitive functions, input and output forms, and defined functions. System commands, system variables and system functions are those consistent with a single user environment. There are no significant design limitations on the rank or shape of arrays or the length of names. The shared variable processor is omitted. Extensions include system functions supporting files of APL arrays. APL equivalents of the BASIC features PEEK, POKE and SYS are included.

Available:
Waterloo Computing Systems Limited
158 University Ave. W.
Waterloo, Ontario
Canada N2L 3E9

Name: **OSI BASIC Enhancer**
System: OSI C1P/ Superboard/C4P
Memory: 8K
Language: Machine code w/BASIC-in-ROM
Hardware: C1P, Superboard, C4P

Description: For the BASIC programmer who wants real power over his stock system. Get real delete action; replace cursor with one of your own choice (defaults to checkerboard square); commands to RENUMBER programs to make them easy to read; AUTOSEQUENCER saves typing in line numbers; screen control command has been added to running BASIC; LOAD and SAVE files w/filenames on a token I/O system reduce load/save times by 50%. Runs in approximately 1.5K of RAM. Send $1.00 for complete catalog.

Price: $19.95 ppd.
Includes autoload, autorun cassette only. Users manual and bug-free guarantee.

Author: Timothy W. Jackson

Available:
Computer Science Engineering
Box 50, 291 Huntington Ave.
Boston, MA 02115

Name: **AIRSIM-1**
System: Apple II or Apple II Plus
Memory: 48K bytes
Language: Machine
Hardware: 1 disk drive, paddles or self-centering joystick, Applesoft in ROM or RAM

Description: *AIRSIM-1* is a realistic simulation of airplane flight. It has scenery from Boston, MA to New York City, with 6 distinct airfields for landings and takeoffs. A score is accumulated for successful landings at three of these fields. *AIRSIM-1* can do loops, rolls and even Immelmann turns. It is equipped for instrument flying, and can make landing approaches on instruments. Instrumentation includes radar, artificial horizon, and horizontal-situation indicator (HSI).

Price: $40.00
Includes diskette and manual

Author: Ted Kurtz

Available:
Mind Systems Corporation
Box 506
Northampton, MA 01061
(413) 586--6463

Name: **Pool 1.5**
System: Apple, Atari
Memory: 48K
Language: Machine
Hardware: Disk II, game paddles

Description: *Pool 1.5* is a real-time, Hi-Res color simulation of pool. This action-packed game allows you to play eight ball, rotation, nine ball, or straight pool.

Price: $34.95

Available:
IDSI
P.O. Box 1658
Las Cruces, NM 88004
(505) 522-7373

Name: **Management System for Stock Control**
System: Apple II
Memory: 48K
Language: Applesoft in ROM
Hardware: Disk and 80- or 132-character per line printer

Description: This inventory management system is designed to offer a complete and current overview of stock with a minimal effort by the operator. Detailed information on any item can be gained instantly. The manual part of the package is written for the novice and comprised of four main sections: Introduction, Practice Run, Reference, and Appendices.

Price: $175.00

Author: JACC, Inc.

Available:
The Hayden Book Company
50 Essex Street
Rochelle Park, NJ 07662

Name: **Color Assembler**
System: TRS-80C Color Computer
Memory: 32K
Language: Assembly
Hardware: TRS-80C

Description: This is a complete 6809 machine code assembler that supports all 6809 mnemonics, addressing codes along with standard assembler options and directives. It operates as a two-pass assembler, so both forward and backward references are allowed. The Motorola Instruction Set Reference Card and documentation on many of the major subroutines in the Color Computer's BASIC are included with the manual.

Price: $29.95
Includes cassette, manual, Reference Card, and BASIC subroutine documentation

Available:
Computerware
P.O. Box 668
Encinitas, CA 92024
(714) 436-3512

(Continued on page 118)

Software Catalog *(continued)*

Name: **Hi-Res/Multi-Color Graphics for BASIC**
System: VIC-20
Memory: 2½K
Language: Assembly
Hardware: Standard VIC-20
Description: These two utilities give the BASIC programmer the ability to use high-resolution and multi-color graphics on a standard VIC without the need to add additional hardware. Hi-Res yields a 104×152 position screen. In multi-color mode you get 52×76 size. You may plot and erase points, lines, boxes, and ASCII text in either Hi-Res or multi-color. All commands available from BASIC programs.
Price: $20.00
Includes manual and sample programs
Author: Roy Wainwright
Available:
Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510
(616) 241-5510

Name: **Hardisk Accounting Software**
Memory: 64K
Language: UCSD Pascal
Hardware: Apple II, Apple III, Corvus or Profile hard disk
Description: *The Hardisk Accounting System* was developed for the company that wants a comprehensive accounting system that can change and grow with them. Until the introduction of the *Hardisk Accounting System*, businesses using microcomputers were limited by the capacity and slow speed of the floppy disk. This program is a menu-driven, double entry accounting system. It consists of general ledger, accounts receivable, accounts payable, inventory, point of sale, sales order entry, purchase order entry, payroll, fixed asset management, and mailing labels. All modules are interactive and include complete audit trails. The business person will find the *Hardisk Accounting System* easy to use, thanks to the data entry prompts and extensive error checking.
Price: $1495.00
Available:
Great Plains Computers
113 Broadway
Fargo, ND 58102

Name: **Descriptive Statistics and Regression Analysis #26011**
System: Apple II, Apple II Plus
Memory: 32K RAM
Language: Applesoft
Hardware: 5¼" disk
Description: This package contains three programs which perform statistical and regression analysis. Included are: Descriptive Statics (mean, standard deviation, variance, kurtosis, z-scores); Curvilinear Regression (linear, inverse, polynomial, exponential, logarithmic); Multivariable Linear Regression.
Price: $39.95
Includes documentation
Available:
Advanced Operating Systems
450 St. John Rd., Suite 792
Michigan City, IN 46360
(219) 879-4693

Name: **VisiFactory**
System: Apple II, Apple II Plus
Memory: 48K
Language: Applesoft in ROM
Hardware: Disk II
Description: Allows a marriage between Data Factory and Visicalc™ files. You can move data in either direction, manipulate it within the chosen program, and then store it either way. It is an exciting tool for market research, information surveys, and analyses of any selected data.
Price: $75.00
Available:
Micro Lab
2310 Skokie Valley Rd.
Highland Park, IL 60035

MICRO

Answer to 6502 Puzzle

The obvious answer, that the program will execute the Jump Indirect through *Vector* and encounter the BRK at address 1000, is WRONG! The 6502 has a slight problem with page boundaries under some conditions. In this example it will perform the Jump Indirect by fetching the low byte of the target address from 6DFF and the high byte of the address from 6D00 — *not* 6E00 as one might expect. The effective address of the instruction will therefore be 6D00 — and the program will loop forever!

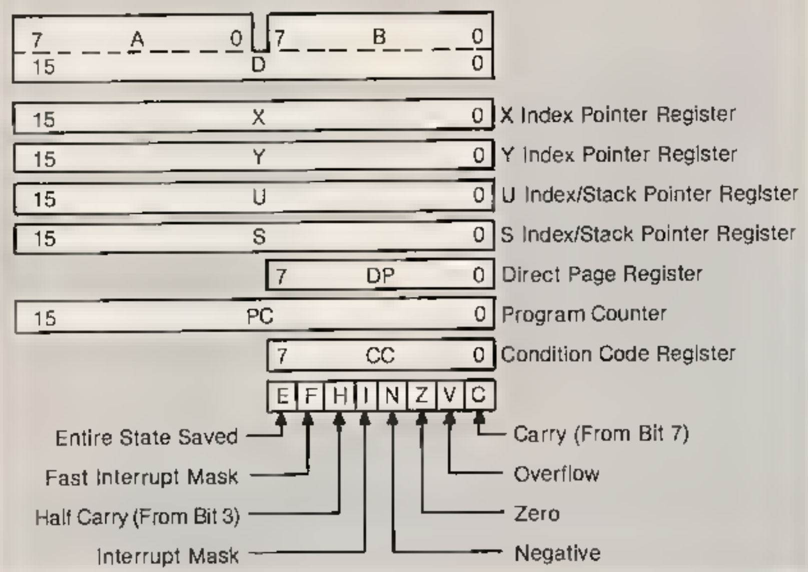
6809 Microprocessor

MC6809/MC6809E—an enhanced 8-bit microprocessor with some 16-bit functions and an 8-bit multiply. It is architecturally similar to both the 6502 and the 6800, and has removed many of their shortcomings. Designed primarily for ease of programming (rather than as a compromise for both programming and dedicated applications), it is particularly desirable for relocatable, reentrant, and modular programming. With its 16-bit capabilities, dual stacks, multiple index registers, and indexing modes, it is good for the stack-oriented implementations of Pascal, FORTH, and other high-level languages.

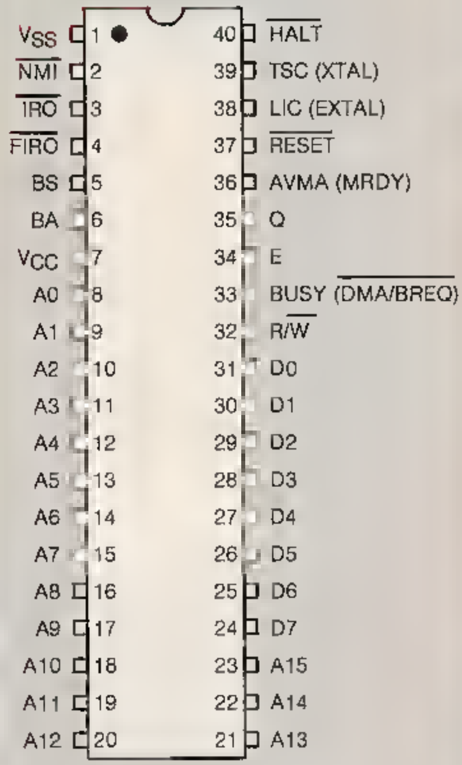
Manufactured by Motorola—The 'E' version requires an external clock and is especially well-suited to multiprocessing applications, such as in Stellation Two's "The Mill" board for the Apple and in Commodore's SuperPET.

Other computers with 6809's are the Radio Shack TRS-80 Color Computer and computers manufactured by Southwest Technical Products, Gimix, The Computerist, Canon, Smoke Signal Broadcasting, Percom Data, and others.

PROGRAMMING MODEL



MC6809E PIN-OUT



MC6809 pin-out in parentheses, where different

INDEXED/INDIRECT CODES

Type	Forms	Non Indirect: Assembler Form	Non Indirect: Postbyte OP Code	Indirect: Assembler Form	Indirect: Postbyte OP Code
Constant Offset From R	No Offset	,R	1RR00100	[,R]	1RR10100
(2's Complement Offsets)	5-Bit Offset	n, R	0RRnnnnn	Defaults To 8-Bit	
	8-Bit Offset	n, R	1RR01000	[n, R]	1RR11000
	16-Bit Offset	n, R	1RR01001	[n, R]	1RR11001
Accumulator Offset From R	A Register Offset	A, R	1RR00110	[A, R]	1RR10110
(2's Complement Offsets)	B Register Offset	B, R	1RR00101	[B, R]	1RR10101
	D Register Offset	D, R	1RR01011	[D, R]	1RR11011
Auto Increment/Decrement R	Increment By 1	,R+	1RR00000	Not Allowed	
	Increment By 2	,R++	1RR00001	[,R++]	1RR10001
	Decrement By 1	,−R	1RR00010	Not Allowed	
	Decrement By 2	,−−R	1RR00011	[,−−R]	1RR10011
Constant Offset From PC	8-Bit Offset	n, PCR	1xx01100	[n, PCR]	1xx11100
(2's Complement Offsets)	16-Bit Offset	n, PCR	1xx01101	[n, PCR]	1xx11101
Extended Indirect	16-Bit Address	—	—	[n]	10011111

R = X, Y, U or S
x = Don't Care

RR:
00 = X
01 = Y
10 = U
11 = S

Instruction	Inherent	Immediate	Direct	Extended	Indexed / Indirect	Relative	E	F	H	I	H	Z	V	C
ABX	3A													
ADCA		89	99	B9	A9				X		X	X	X	X
ADCB		C9	D9	F9	E9				X		X	X	X	X
ADDA		8B	9B	BB	AB				X		X	X	X	X
ADDB		CB	DB	FB	EB				X		X	X	X	X
ADDD		C3	D3	F3	E3						X	X	X	X
ANDA		84	94	B4	A4						X	X	0	
ANDB		C4	D4	F4	E4						X	X	0	
ANDCC		1C												
ASL			08	78	68				X		X	X	X	X
ASLA	48								X		X	X	X	X
ASLB	58								X		X	X	X	X
ASR			07	77	67				X		X	X		X
ASRA	47								X		X	X		X
ASRB	57								X		X	X		X
BITA		85	95	B5	A5						X	X	0	
BITB		C5	D5	F5	E5						X	X	0	
CLR			0F	7F	6F						0	1	0	0
CLRA	4F										0	1	0	0
CLRB	5F										0	1	0	0
CMPA		81	91	B1	A1				X		X	X	X	X
CMPB		C1	D1	F1	E1				X		X	X	X	X
CMPD		10 83	10 93	10 B3	10 A3						X	X	X	X
CMPS		11 8C	11 9C	11 BC	11 AC						X	X	X	X
CMPU		11 83	11 93	11 B3	11 A3						X	X	X	X
CMPX		8C	9C	BC	AC						X	X	X	X
CMPY		10 8C	10 9C	10 BC	10 AC						X	X	X	X
COM			03	73	63						X	X	0	1
COMA	43										X	X	0	1
COMB	53										X	X	0	1
CWAI		3C												
DAA	19										X	X	X	X
DEC			0A	7A	6A						X	X	X	
DECA	4A										X	X	X	
DECB	5A										X	X	X	
EORA		88	98	B8	A8						X	X	0	
EORB		C8	D8	F8	E8						X	X	0	
EXG		1E												
INC			0C	7C	6C						X	X	X	
INCA	4C										X	X	X	
INCB	5C										X	X	X	
JMP			0E	7E	6E									
JSR			9D	BD	AD									
LDA		86	96	B6	A6						X	X	0	
LDB		C6	D6	F6	E6						X	X	0	
LDD		CC	DC	FC	EC						X	X	0	
LDS		10 CE	10 DE	10 FE	10 EE						X	X	0	
LDU		CE	DE	FE	EE						X	X	0	
LDX		8E	9E	BE	AE						X	X	0	
LDY		10 8E	10 9E	10 BE	10 AE						X	X	0	
LEAS					32									
LEAU					33									
LEAX					30							X		
LEAY					31							X		
LSL			08	78	68				X		X	X	X	X
LSLA	48								X		X	X	X	X
LSLB	58								X		X	X	X	X
LSR			04	74	64						0	X		X
LSRA	44										0	X		X
LSRB	54										0	X		X

Instruction	Inherent	Immediate	Direct	Extended	Indexed / Indirect	Relative	E	F	H	I	H	Z	V	C
MUL	3D											X		X
NEG			00	70	60				X		X	X	X	X
NEGA	40								X		X	X	X	X
NEGB	50								X		X	X	X	X
NOP	12													
ORA		8A	9A	BA	AA						X	X	0	
ORB		CA	DA	FA	EA						X	X	0	
ORCC		1A												
PSHS		34												
PSHU		36												
PULS		35												
PULU		37												
ROL			09	79	69						X	X	X	X
ROLA	49										X	X	X	X
ROLB	59										X	X	X	X
ROR			06	76	66						X	X		X
RORA	46										X	X		X
RORB	56										X	X		X
RTI	3B													
RTS	39													
SBCA		82	92	B2	A2				X		X	X	X	X
SBCB		C2	D2	F2	E2				X		X	X	X	X
SEX	1D										X	X		
STA			97	B7	A7						X	X	0	
STB			D7	F7	E7						X	X	0	
STD			DD	FD	ED						X	X	0	
STS			10 DF	10 FF	10 EF						X	X	0	
STU			DF	FF	EF						X	X	0	
STX			9F	BF	AF						X	X	0	
STY			10 9F	10 BF	10 AF						X	X	0	
SUBA		80	90	B0	A0				X		X	X	X	X
SUBB		C0	D0	F0	E0				X		X	X	X	X
SUBD		83	93	B3	A3						X	X	X	X
SWI	3F						1	1						
SWI2	10 3F						1							
SWI3	11 3F						1							
SYNC	13													
TFR		1F												
TST			0D	7D	6D									
TSTA	4D										X	X	0	
TSTB	5D										X	X	0	

Branch Instruction Codes

BCC	24	LBCC	10 24	BLT	2D	LBLT	10 2D
BCS	25	LBCS	10 25	BMI	2B	LBMI	10 2B
BEQ	27	LBEQ	10 27	BNE	26	LBNE	10 26
BGE	2C	LBGE	10 2C	BPL	2A	LBPL	10 2A
BGT	2E	LBGT	10 2E	BRA	20	LBRA	16
BHI	22	LBHI	10 22	BRN	21	LBRN	10 21
BHS	24	LBHS	10 24	BSR	8D	LBSR	17
BLE	2F	LBLE	10 2F	BVC	28	LBVC	10 28
BLO	25	LBLO	10 25	BVS	29	LBVS	10 29
BLS	23	LBLS	10 23				

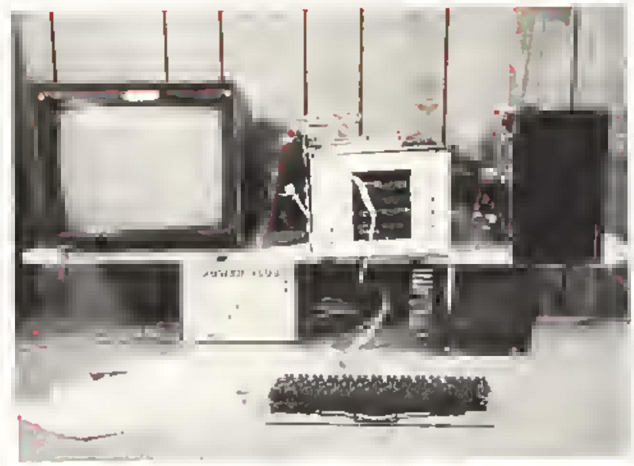
It's All 1's and 0's

No Connection

MICRO has maintained its subscription information for several years on a KIM-based computer system with a "homebrew" floppy disk controller. Last spring MICRO's sister company, The Computerist, decided to make a product that would include the floppy disk controller. A few minor design improvements were made and the board was sent out for PC layout. When the prototype board had been made and assembled, I prepared to test it. I expected to take an hour or two to test and verify the new version. It took several long days! Nothing seemed to work right. Even though each and every signal to the disk seemed to be okay, the system would not work. I set up a working system and compared it step-by-step with the new system and could find no difference on any of the control or data signals. I tried changing the various IC chips in the circuit and found that different floppy disk controller chips gave different results (this design used the popular Western Digital 1791). One chip would cause the drive to step in and step out on command but could not successfully perform a seek; another chip would restore and seek on track 00 but would not step at all; another would do nothing. Very strange and very frustrating. How could the identical design not work?

I had noticed, on one of my many examinations of the connections to the 1791, that there was a ground connection to a pin marked "No Connection." I had dismissed this as a possible cause of the problem, reasoning that this unused pin had no internal connection and was there simply to be pin 40 of the IC package. Having run out of sensible things to try, I finally cut the ground connection. Surprise — that cured the problem! What I had not known, at the time, was that "No Connection" did *not* mean that there was no connection to this pin on the IC itself, but that no connection should be made to this pin. Why? Because there *is* a connection to this pin within the 1791 chip. This pin is used in the manufacture and/or testing of the 1791 and must be left unconnected.

Doctor Bob



The Original MICRO KIM-Based Subscription System (believe it or not!)

The New MICRO FOCUS-Based Subscription System (The Computerist FOCUS System)

A 6502 Puzzle

Here is a little puzzle about the 6502. Since it is only two instructions and three lines of code, it can't be that tough, can it?

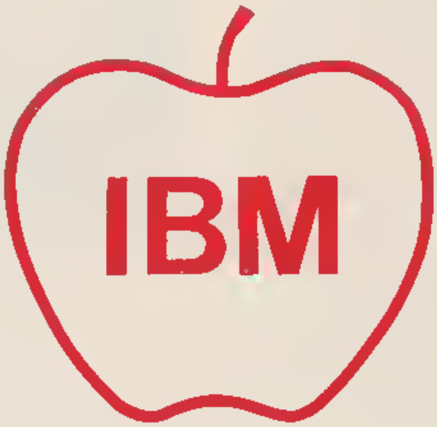
```
1000    00                      BRK
6D00    6D 6D FF    START       JMP     (VECTOR)
6DFF    00 10       VECTOR      =       $1000
```

You can assume that the interrupts have been properly set up so that a BRK will go to a monitor. The simple question is, when this program is started at START, what will happen?

(Based on a note from Earl Morris of Midland, Michigan)

For answers to 6502 puzzle, see page 118.

Please send your unusual observations, puzzles, programming tricks, system photos, etc. to Dr. Bob, MICRO, P.O. Box 6502 Chelmsford, MA 01824



Write IBM® on your Apple.®

Our UCSD p-System® lets you write applications on your Apple II that will run without recompilation on the IBM Personal Computer.

And that's not all.
Your UCSD p-System applications will also run on Commodore, Philips, Radio Shack and Texas Instruments — in fact, on just about any popular microcomputer you can name. Because SofTech Microsystems' UCSD p-System is the only microprocessor-independent operating system for 8-bit and 16-bit computers.

Our enhanced UCSD p-System gives you everything you need to develop genuinely portable, compact applications for almost every microcomputer owner. It allows you to work in any combination of UCSD Pascal,® BASIC and assembly language. And it provides support for dynamic memory management and multitasking, with a full arsenal of enhancements such as a powerful Screen Editor, Native Code Generators, TURTLEGRAPHICS and XenoFile,® for easy access to CP/M® disks and files.

Our UCSD p-System is available for distribution licensing as well as single copy purchase. Visa and MasterCard orders are welcome.

Write to us for more details.
SofTech Microsystems, Inc.
9494 Black Mountain Road,
San Diego, CA 92126.
Or call (714) 578-6105,
TWX: 910-335-1594.

UCSD p-System and UCSD Pascal are trademarks of the Regents of the University of California. XenoFile is a trademark of SofTech Microsystems, Inc. Apple is a registered trademark of Apple Computer, Inc. IBM is a registered trademark of International Business Machines Corporation. CP/M is a registered trademark of Digital Research, Inc.

SOFTECH
MICROSYSTEMS
A SUBSIDIARY OF SOFTECH

MICRO

Advertiser's Index

Next Month in MICRO

May PET Feature

- **PET Menu and Tape Timer** — This article describes a menu program that allows rapid access to any program on either side of a cassette tape. In addition, a tape timer is presented that supplies the fast forward timer for the menu program. These two programs feature advanced cassette control and use the WAIT command extensively.
- **Growing Knowledge Trees** — Knowledge often can be represented in tree diagrams. Microcomputers can store and analyze these diagrams. This PET program finds out what people know about a topic, analyzes answers, and shows users the organized results. A BASIC and an assembly language routine are presented for analyzing the diagrams.
- **PET Memory Protector** — Allows PETs with static RAM to protect 1K or more from resets, LOADs, and BASIC, by inserting a circuit between a RAM chip and its socket.

Regular Columns

From Here to Atari
PET Vet
The Single Life

Other May Feetures

LISZT with Strings for the Apple
AID Conversion Using a 555 Timer IC for the Atari
Apple Graphics for Okadata Microline 80
A General BASIC — Machine Language Interface for the AIM
Memory Map Relocater for the Apple

1982 Editorial Schedule

June — Applications
July — Apple
August — Programming Techniques
September — 68000
October — Education
November — Atari/Games
December — PET

20% OFF

Your money goes farther when you subscribe. During the course of a year, when you subscribe, you save 20% (in the U.S.).

Pay only $24.00 ($2.00 a copy) for 12 monthly issues of MICRO sent directly to your home or office in the U.S.

More MICRO for Less Money When You Subscribe

But on the newsstand — if you can locate the issue you want — you pay $30.00 a year ($2.50 a copy).

Special Offer — Subscribe for 2 years ($42.00) and get 30% off the single issue price.

Subscribe to MICRO today.

MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824

Please send me MICRO for __ 1 year __ 2 years
NOTE: Airmail subscriptions accepted for 1 year only.

Check enclosed $________
Charge my ________ VISA account
________ Mastercard account

No. ______________________

Expiration date ______________________

Name ______________________

Address ______________________

____

City/State ______________________ Zip ____

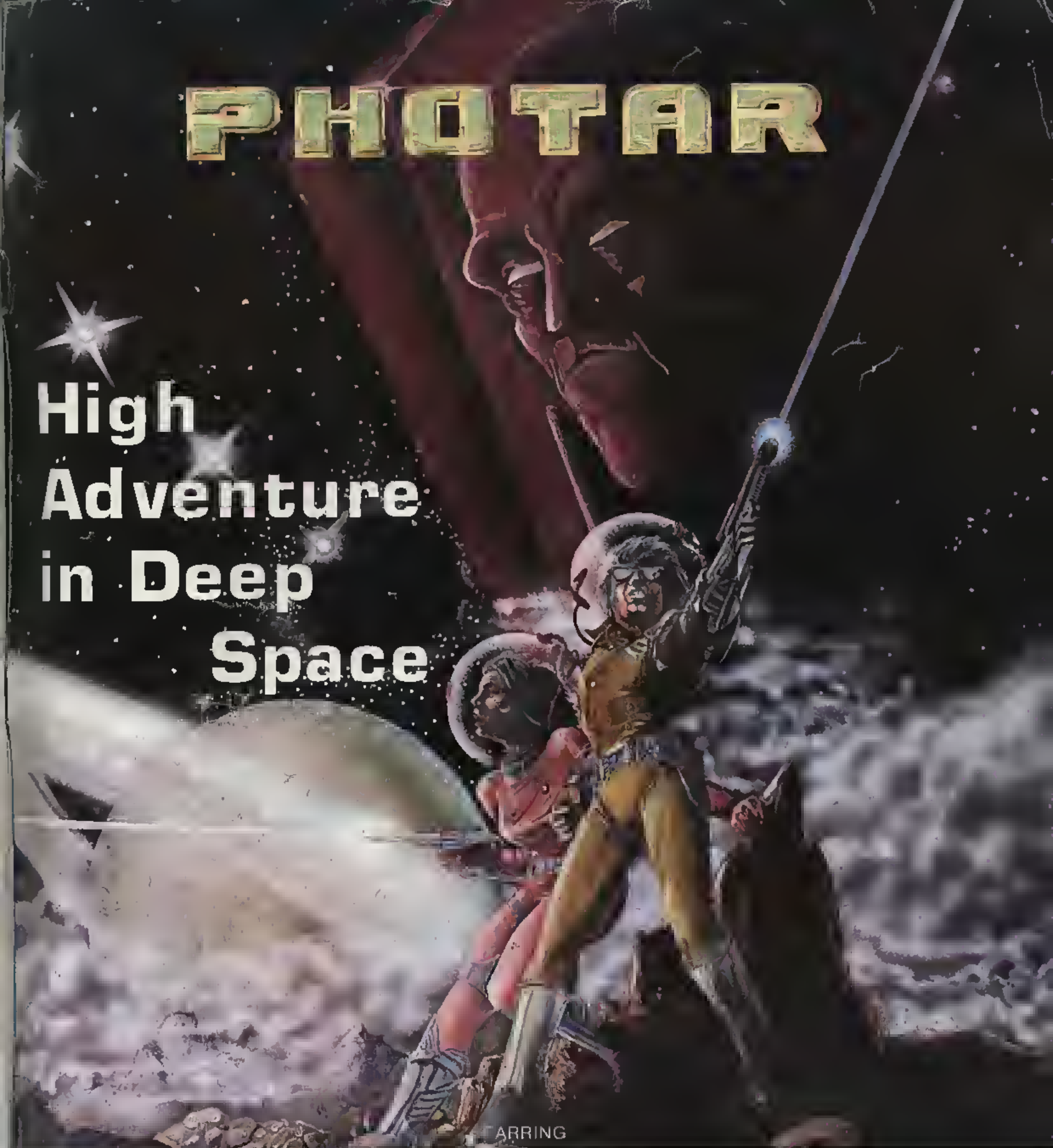
Subscription Retes Effective Jenuery 1, 1982

Country	Rete
United States	$24.00 1 yr.
	42.00 2 yr.
Foreign surface mail	27.00
Europe (air)	42.00
Mexico, Central America, Mid East, N. & C. Africa	48.00
South Am., S. Afr., Far East, Australasia, New Zealand	72.00

* Airmail subscriptions accepted for only 1 year. For U.S. and Canadian 2-year rates, multiply by 2.

Job Title: ______________________

Type of Business/Industry: ______________________



PHOTAR ■ THE RINGS OF SATURN ■ PULSARS ■ KILOSTARS ■ BLACK STARS

IN

The Ultimate Experience in Outer Space Adventure
The Ultimate Experience in Outer Space Adventure
The Ultimate Experience in Outer Space Adventure

Produced by SOFTAPE
An Artsci Company

Screenplay by SAB

Distributed through Softape, Softsel, and other computer stores world-wide

10432 BURBANK BOULEVARD ■ NORTH HOLLYWOOD, CALIFORNIA 91601 ■ 213/985-5763